UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS (CC)

MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO - UFC

DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL SIQUEIRA TELLES VIEIRA

USO DE SUBÁRVORES ESPECIALIZADAS PARA OBTENÇÃO DE DESFAZER E TUTORIAL LOCALIZADOS EM SISTEMAS DE CONTROLE DE VERSÃO DE MODELOS 3D

FORTALEZA

2016

RAFAEL SIQUEIRA TELLES VIEIRA

USO DE SUBÁRVORES ESPECIALIZADAS PARA OBTENÇÃO DE DESFAZER E TUTORIAL LOCALIZADOS EM SISTEMAS DE CONTROLE DE VERSÃO DE MODELOS 3D

Tese apresentada ao Curso de Doutorado em Ciência da Computação do Mestrado e Doutorado em Ciência da Computação - UFC do Centro de Ciências (CC) da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Computação Gráfica

Orientador: Prof. Dr. Joaquim Bento Cavalcante-Neto

Co-Orientador: Prof. Dr. Creto Augusto Vidal

FORTALEZA

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

V718u Vieira, Rafael Siqueira Telles.
Uso de Subárvores Especializadas para Obtenção de Desfazer e Tutorial Localizados em Sistemas de Controle de Versão de Modelos 3D / Rafael Siqueira Telles Vieira. – 2016.
122 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2016.
Orientação: Prof. Dr. Joaquim Bento Cavalcante-Neto.
Coorientação: Prof. Dr. Creto Augusto Vidal.

1. Controle de Versão 3D. 2. Subárvores Especializadas. 3. Desfazeres Localizados. I. Título.
CDD 005

RAFAEL SIQUEIRA TELLES VIEIRA

Uso de Subárvores Especializadas para Obtenção de Desfazer e Tutorial Localizados em Sistemas de Controle de Versão de Modelos 3D

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito para a obtenção do grau de Doutor em Ciência da Computação.

Aprovado em 30 de agosto de 2016.

BANCA EXAMINADORA

Prof. Dr. Joaquim Bento Cavalcante Neto
(Presidente)
Universidade Federal do Ceará - UFC

Prof. Dr. Creto Augusto Vidal
(Coorientador)
Universidade Federal do Ceará - UFC

Prof. Dr. William Wagner Matos Lira
Universidade Federal de Alagoas - UFAL

Prof. Dr. Evandro Parente Júnior
Universidade Federal do Ceará - UFC

Prof.ª Dr.ª Emanuele Marques dos Santos
Universidade Federal do Ceará – UFC

A Deus.

A minha família.

AGRADECIMENTOS

Agradeço a Deus, por ter tido a força de espírito para concluir esta tese, e por ser meu apoio e rochedo em todos os momentos de angústia e sofrimento. Sem ele jamais teria tido a capacidade de concluí-lo, pois são muitos os que plantam dúvidas e poucos os que promovem a esperança.

Agradeço em seguida a minha esposa, Rossana, pela compreensão nesta tese e seu apoio incomensurável. Não há palavras suficientes para lhe agradecer.

Agradeço também meu padrinho, Francisco José Siqueira Telles, que me ajudou dando a injeção de ânimo e forças que precisava no momento certo, além do seu exemplo de pessoa e caráter raríssimo nos dias de hoje.

Agradeço meus pais, Clotilde e Carlos, pela insistência no meu estudo e capacitação quando ainda era novo que me motivou a sempre perseguir patamares cada vez mais altos, e pelo exemplo que me deram ao longo da vida.

Agradeço meus irmãos de sangue, Ricardo, Carla e Carlos Alberto, e meus irmãos de coração, Gustavo Wilke, Hister Lima e Ricardo Brasil, pelo apoio e companhia ao longo de todo o processo de construção desta tese.

Agradeço meus orientadores, Joaquim Bento e Creto Vidal, pela confiança e apoio ao meu trabalho. Por me oferecerem a chance de escrever esta tese e cursar o doutorado. Pela viagem a Nova Iorque e a possibilidade de desenvolver na NYU-Poly parte desta tese, tendo contato com outros pesquisadores e professores. Também sou profundamente grato a eles, como Guillaume Vialaneix e Cláudio T. Silva, que me auxiliaram nesta pesquisa através de seus apontamentos e orientação, e os meus colegas de quarto, Guilherme Oliveira e Fábio Markus Miranda, cujo apoio nesta época difícil foi essencial.

Agradeço a todos os pesquisadores que distribuem seus artigos gratuitamente sob o formato digital o que facilitou enormemente a realização desta tese. Muitos deles estão citados nas referências. Pelo desenvolvimento desta pesquisa que só pode ser feita pelo trabalho conjunto de tantos outros estudiosos que mantêm a ciência livre do papel moeda.

Por fim, agradeço a CAPES pelo fomento de minha pesquisa ao longo dos anos e minha ida a Nova Iorque.

A todos, o meu mais sincero e verdadeiro muito obrigado! Esta tese só foi possível graças a todos vocês.

“Esforçai-vos, e ele fortalecerá o vosso coração,
vós todos que esperais no Senhor."

(Bíblia, Salmos 31:24.)

RESUMO

Um sistema que grava mudanças realizadas nos dados de um arquivo é chamado de Sistema de Controle de Versões. Nesse tipo de sistema, cada versão corresponde a uma mudança efetuada. Sistemas de controle de versões podem armazenar todo tipo de arquivos, contudo o foco desta tese é em mudanças feitas sobre arquivos de malhas poligonais, i.e., modelos tridimensionais. A estrutura de dados usada por esses sistemas são árvores ou grafos direcionados acíclicos, sendo que esta tese foca em árvores de controle de versão (ACVs). O método desenvolvido permite que o usuário rastreie o histórico de mudanças geométricas e topológicas que ocorreram sobre uma determinada região da malha para desfazer algumas das operações efetuadas ou para desenvolver um tutorial. Para efetuar esse rastreamento, a região desejada é selecionada através de uma Caixa Envoltória, que é propagada para todos os nós da ACV, de modo a encontrar as interseções da seleção com os Conjuntos de Mudanças, i.e., encontrar as diferenças entre duas versões consecutivas de uma sequência de modelagem. Para encontrar esses conjuntos, é feito um mapeamento de elementos entre versões vizinhas usando-se de propagação topológica e tabelas de dispersão (*hash*), que são responsáveis respectivamente por avançar a fronteira entre elementos mapeados e não mapeados de uma malha, e desenvolver uma chave única para elementos que não se alteram entre versões consecutivas. Com essas informações, é possível extrair uma subárvore associada com a região selecionada de maneira a servir como um tutorial sobre como aquela região foi desenvolvida ao longo do tempo para fins de aprendizagem. Essa subárvore também permite desfazer operações locais, i.e., operações que não mudam todos os elementos de uma versão da malha, de modo a poupar tempo de modelagem, que normalmente é alto mesmo para malhas simples. Nesta tese, é desenvolvida uma operação especialista de desfazer que atua apenas sobre uma região específica, chamada de Desfazer Localizado, que é uma das principais contribuições desta tese. As limitações do método, os meios para superá-las, e os resultados obtidos pela técnica em seu estado são discutidos e apresentados. A técnica é comparada com outra que pertence ao estado da arte, e é também analisada com relação a eficiência de geração de chaves únicas, i.e, a perfeição do índice de dispersão, e quanto a sua velocidade que representam dois pontos importantes do algoritmo, pois lida com grandes conjuntos de dados.

Palavras-chave: Controle de Versão 3D. Subárvores Especializadas. Desfazeres Localizados.

ABSTRACT

A system that records changes made to a file is called a Version Control System (VCS). Even though VCSs may store all kinds of files, our thesis focus on changes made to polygonal mesh files. The data structure used by those systems are trees or directed acyclic graphs, and our work focuses on version control trees (VCTs). Our method allows the user to track the history of topological and geometrical changes made to a mesh region to generate a tutorial or to undo some operations made to that region. The region to be tracked is selected through a bounding box selection mechanism, which is propagated to all VCT nodes so we can find the intersection sets between the bounding-box selection and *Changesets*, i.e., the difference between two consecutive versions. To identify the *Changesets*, the algorithm maps elements from one version to another using topological propagation and hash tables, which are responsible for propagating the border between mapped and unmapped elements, and to create a unique key for elements that do not change between consecutive versions. With these data, it is possible to construct a subtree associated with the selected region to serve as a tutorial on how that region was modeled. That subtree also allows the user to undo local changes, i.e., operations that do not change all elements of a mesh version, to save modeling time, which is normally high even for small meshes. This undo operation acts just over a specific region and it is a specialized undo, as such, we call it bounded undo, which is an important contribution in this Thesis. Despite the important contributions, we also point out some current limitations to our method and discuss ways to overcome them. Our technique is also compared with the current state of the art version control systems for meshes, and is analyzed with respect to efficiency of unique key generation and speed, which are two important points of the algorithm, since it deals with large datasets.

Keywords: 3D Version Control. Specialized Subtrees. Bounded Undo.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*/interface de aplicação do programa
CAD	*Computer Aided Design*/projeto auxiliado por computador
CSG	*Constructive Solid Geometry*/geometria de construção de sólidos
DAG	*Directed Acyclic Graph*/grafo direcionado acíclico
DBL	*Demonstration Based Learning*/ensino baseado em demonstração
HT	*Hyper-Threading*/hiperprocessamento
MEL	*Maya Embedded Language*/linguagem intríseca do Maya
NUMA	*Non-uniform memory access*/acesso não-uniforme da memória
RAM	*Random Access Memory*/memória de acesso aleatório
RGBD	*Red Green Blue Depth*/informações de câmera: vermelho, verde, azul e profundidade
RGB	*Red Green Blue*/padrão de cores: vermelho, verde e azul
XML	*eXtensible Markup Language*/linguagem de marcação extendida
2D	duas dimensões
3D	três dimensões
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ACV	Árvore de Controle de Versão
BD	Bando de Dados
GB	10^9 Bytes
GHz	10^9 Hertz
MQ	Socketed Quad Core
NP	Não Polinomial
SCV	Sistema de Controle de Versão
VT	Vistrails

LISTA DE SÍMBOLOS

$alfabeto\_minúsculo_{grego}$ elemento unidimensional

$alfabeto\_minúsculo_{romano}$ elemento tridimensional

$alfabeto\_maiúsculo_{romano}$ conjunto de elementos ou conjunto de conjuntos

$\vec{a}$	vetor *a* tridimensional com coordenadas reais
Λ	Conjunto das partes
$\forall$	Para todo
$\exists$	Existe
$\leftarrow$	Atribuição
$\subset$	Está contido
$\supset$	Contém
$\in$	Pertence
$\vee$	E lógico
$\wedge$	Ou lógico
$\leqslant$	Menor ou igual
$<$	Menor
$\geqslant$	Maior ou igual
$>$	Maior
$\cap$	Intersecção
$\neq$	Diferente
$\perp$	Perpendicular
$=$	Igual
$\emptyset$	Conjunto vazio
$\bigcup$	União de conjuntos
A_t	Matriz na versão t
B	Árvore de versões
BB	*Bounding Box*/Caixa Envoltória

B_{sub}	Subárvore especialista de versões
D	Deleção
E_t	Extrusão de aresta na versão t
F_t	Face ou inserção de face na versão t
I	Inserção
M_t	Malha na versão t
$\mathbb{N}$	Conjunto dos números naturais
$proj$	Projeção
OP	Operação
OP^{-1}	Inversa da matriz contida na operação
$\mathcal{O}$	Ordem de complexidade
R	Região
$\mathbb{R}$	Conjunto dos números reais
S	Seleção
T	Função de transformação
U	Conjunto universo
$\mathbb{V}$	Versão
V_t	Vértice na versão t

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo são detalhados a motivação do trabalho (Seção 1.1), o problema a que buscou-se responder (Seção 1.2), os objetivos a serem alcançados para resolução do problema (Seção 1.3), a estratégia adotada para alcançar os objetivos (Seção 1.4), e a contribuição obtida com o desenvolvimento da técnica (Seção 1.5). Além disso, na seção final, é descrito como esta tese se organiza em capítulos (Seção 1.6).

1.1 Motivação

Modelos tridimensionais digitais são muito usados atualmente em diversos ramos da indústria, e podem ser obtidos de diversas maneiras: por sobreposição de imagens com Estereoscopia, ou Holografia, por uma nuvem de pontos tridimensionais obtida com um scanner 3D, e por Modelagem através de um artista que cria estruturas digitais advindas de sua imaginação ou do mundo real (Figura 1.1).

O foco desta tese é o processo de modelagem feito por artistas, e que pode ser realizado por duas maneiras em prática atualmente: *Drawing* (desenho), que é o desenho manual com interação direta com os elementos geométricos de um modelo, e *Sculpting* (escultura), que é a modelagem feita de forma indireta, i.e., o artista não interage diretamente com a geometria de um modelo. Foca-se, portanto, na modelagem feita de forma direta, que é a primeira que surgiu e ainda é muito utilizada atualmente.

Artistas desenham modelos 3D que podem tomar horas de trabalho, conforme demonstra-se na Tabela 1.1. Logo, quaisquer recursos que possam ser desenvolvidos de forma a otimizar este trabalho, reduzindo o tempo ou esforço para realizar determinadas operações dentro de um modelador, é um avanço significativo para a área.

Artistas também costumam trabalhar em equipes que não precisam estar situadas na

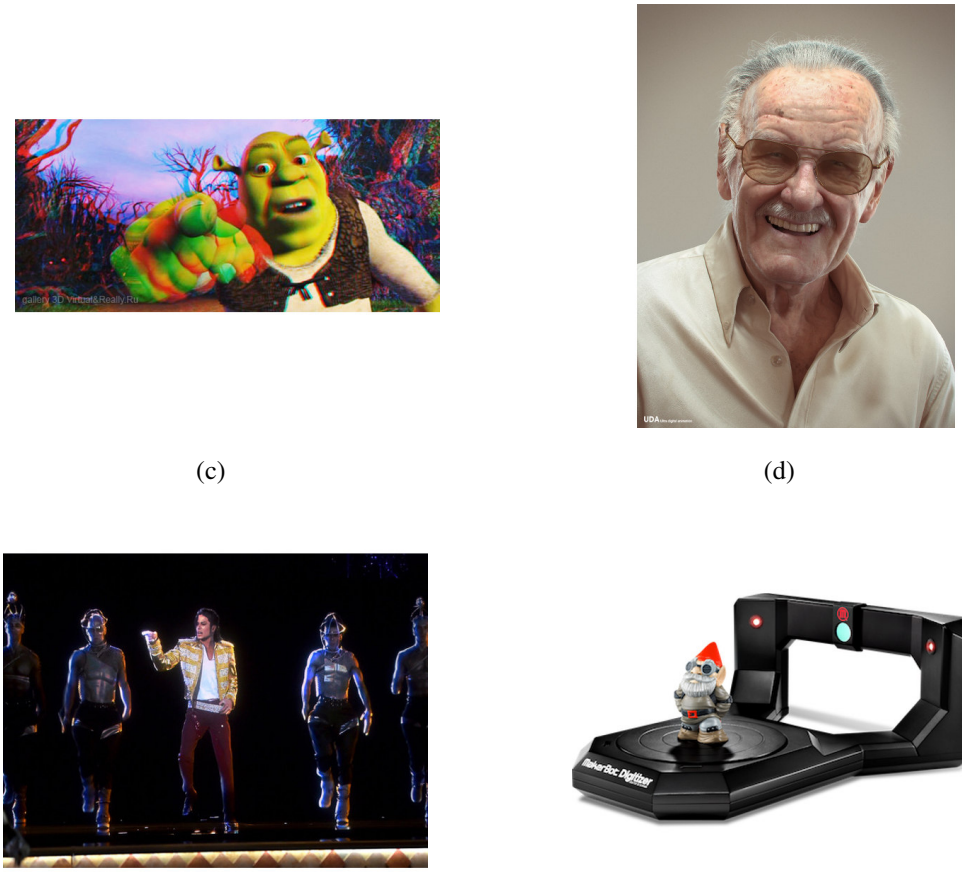
Tabela 1.1 – Tempo para construção de modelos tridimensionais.

Modelo	Vértices	Tempo de Modelagem
Helmo	1342	5h05m
Tubarão	940	3h30m
Hidrante	10435	2h30m
Bípede	564	3h10m
Robô	16081	9h40m

Fonte: Denning *et al.* (2011).

Figura 1.1 – Várias formas para produção de modelos tridimensionais: (a) Estereoscopia usada no filme Shrek de 2001 da Dreamworks, (b) Arte digital com fotorrealismo de Stan Lee feita por modelagem do artista Andrew Hakim de 2012, (c) Aparição de Michael Jackson usando Holografia no Billboard Music Awards de 2014, e (d) Scanner 3D da MakerBot usado para gerar modelos tridimensionais.

(a) (b)



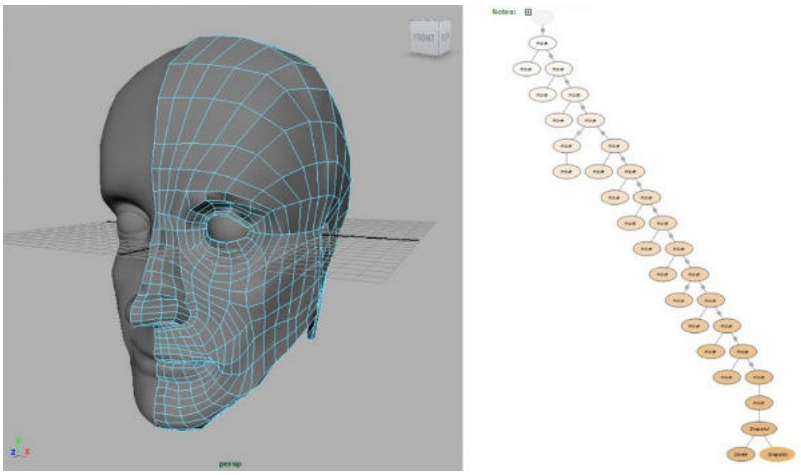
(c) (d)

Fonte: (a) DreamWorks (2001); (b) Hakim (2012); (c) Winter (2014);(d) MakerBot (2016).

mesma parte do globo. Nos últimos anos, houve o crescimento no uso de sistemas integrados (com computadores, tablets, celulares, e televisões digitais conectados uns aos outros), assim como a demanda por ferramentas que permitam trabalho colaborativo e remoto. Sistemas de controle de versões (SCV) surgem para atender esta necessidade, e para permitir o trabalho colaborativo e remoto entre artistas em um produto comercial, e.g, como em um filme (HUBERT, 2013), ou entre pesquisadores (GIRARD *et al.*, 2013) sem restrições geográficas posicionais.

Um sistema de controle de versões armazena informações sobre mudanças que ocorrem em determinados tipos de arquivos (um conjunto de dados). Quando um arquivo sofre uma mudança, diz-se que ele se encontra em uma nova versão. A estrutura de dados utilizada por esses sistemas é uma árvore ou um grafo direcionado acíclico. Como o sistema adotado para

Figura 1.2 – Sistemas de controle de versões podem ser usados durante um processo de modelagem através de plugins em um modelador, como o Maya por exemplo.



Fonte: N.Y.U. (2009).

Nota: Cada operação realizada pelo artista no modelo tridimensional da cabeça é salva como um nó de uma árvore de versões.

controle de versões usado por esta tese baseia-se em árvores (Figura 1.2), este é o principal meio abordado ao longo de todo o texto.

As mudanças feitas em um conjunto de dados quaisquer, assim como a autoria das mudanças (o autor, a hora e a data) são armazenadas por sistemas de controle de versões. Nestes sistemas, também é possível reverter algumas das mudanças feitas sobre um conjunto de dados. Essa operação de reverter as mudanças em um conjunto de dados é conhecida como desfazer. A operação de desfazer (Figura 1.3) possui as seguintes características:

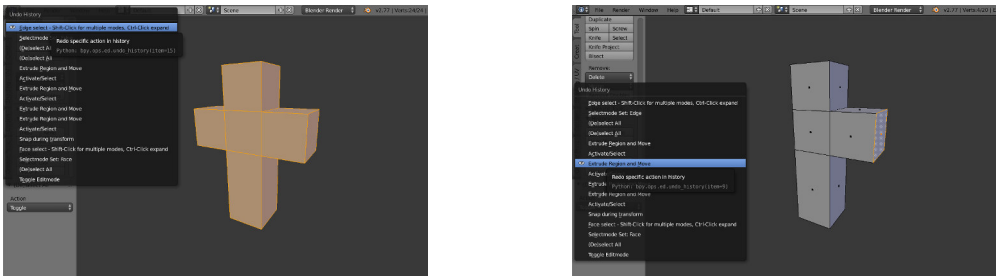
a) Atua linearmente no tempo de modelagem;
b) Não permite qualquer tipo de seleção; e
c) Não permite remover mudanças preservando outras.

Sistemas de controle de versões para textos e imagens permitem desfazer mudanças com restrições espaciais através de uma operação de deleção. Contudo, texto e imagem possuem uma estrutura espacial uniforme (os *pixels* de uma imagem representam um plano bidimensional limitado, e um texto representa um vetor ou matriz de caracteres), enquanto modelos tridimensionais podem apresentar formas geométricas variadas. Isto limita o que pode ser desfeito nestes modelos, pois a ordem das mudanças realizadas influencia o resultado final. Além disso, mesmo que se recorra à operação de deleção, pode-se perder mais mudanças do que se deseja.

O foco desta tese é desfazer mudanças em modelos tridimensionais armazenados em sistemas de controle de versões por meio de árvores. Contudo, quando se atravessa uma

Figura 1.3 – A construção de um modelo tridimensional pode ser encarada como um filme em que cada versão representa um quadro ou uma foto de sua construção.

(a) Versão 15 (b) Versão 9

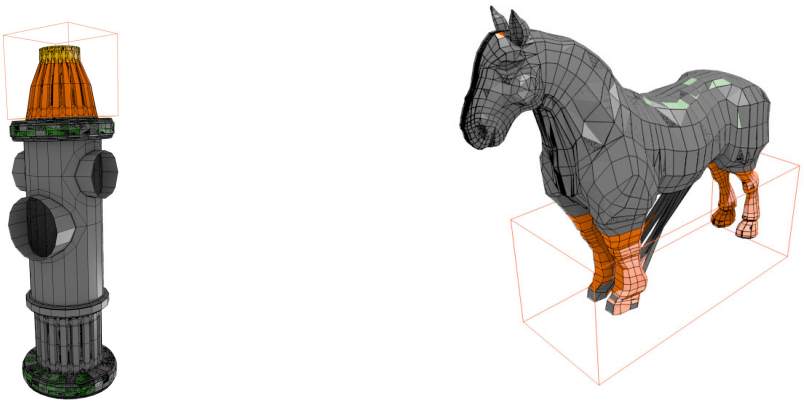


Fonte: o autor.

Nota: O desfazer de operações é equivalente a retroceder no filme da construção do modelo. A parte (a) da figura exibe a versão 15 do modelo, e a parte (b) da figura exibe a versão 9 do modelo; foram desfeitas as operações das versões 10, 11, 12, 13, 14 e 15 para chegar à versão 9.

Figura 1.4 – Exemplos de problemas encontrados durante o processo de modelagem: a parte (a) da figura faces duplicadas aparecem como parte do conjunto de mudanças em verde, quando de fato a mudança ocorreu no topo do modelo do Hidrante em amarelo; e, na parte (b) da figura, erro cometido pelo artista no modelo do Cavalo, quando criou faces que ligam a barriga do cavalo às patas, onde essas faces não devem ser mapeadas.

(a) (b)



Fonte: o autor.

árvore de controle de versões de um modelo tridimensional, é possível encontrar números de faces e vértices variados, conectividade diversa, elementos sobrepostos ou duplicados, estruturas *non-manifold*, etc. (Figura 1.4). Essas ocorrências não são uma limitação *per se* para uma operação de desfazer, mas quando se realiza um desfazer modificado, conforme é feito nesta tese, é preciso leva-las em consideração.

Outro fator a ser observado é que árvores de controle de versão para modelos tridimensionais crescem rapidamente com o tempo, impondo desafios para exibição e manipulação geométrica das mesmas, i.e., árvores grandes impõem problemas de armazenamento e velocidade

de travessia.

Há duas maneiras comumente e atualmente utilizadas para armazenar mudanças em modelos tridimensionais em sistemas de controle de versões: ou o modelo inteiro é armazenado a cada mudança efetuada, o que gera problemas de espaço de armazenamento, ou as funções API de um modelador são armazenadas, o que gera problemas na velocidade de processamento (os modelos são gerados em tempo de execução). O método utilizado nesta este é o de armazenamento de funções API, porém, com a utilização de meios para acelerar esse procedimento, os quais são discutidos em detalhe.

Neste momento, é importante destacar que um histórico de comandos não é um sistema de controle de versões, mas uma pilha limitada dos últimos comandos realizados dentro de um modelador 3D. Um sistema de controle de versões possui tr^ês características básicas (RAYMOND, 1992): reversibilidade, que é a capacidade de reverter mudanças feitas; concorrência, que é o meio para que mais de uma pessoa trabalhe ao mesmo tempo em um conjunto de dados (com controle de sessão e de mesclagem de dados manual ou automática); e anotação, que é a inserção de comentários sobre uma mudança efetuada. Contudo, um histórico de comandos pode ser evoluído para um sistema de controle de versões, pois podem compartilhar informações comuns.

1.2 Problema

Artistas quando atuam em modelagem tridimensional podem efetuar diversos tipos de operações, entre elas está o desfazer de mudanças realizadas em um modelo. Conforme já foi visto, criar tais modelos pode tomar bastante tempo de um artista, e as mudanças geométricas e topológicas que ocorrem durante o processo de modelagem podem ser armazenadas em um sistema de controle de versões através de uma árvore de versão. O problema, então, que esta tese busca solucionar é: como aperfeiçoar o desfazer de mudanças para facilitar o trabalho de artistas no processo de modelagem utilizando-se de uma árvore de controle de versões?

Responder a esta questão amplia a capacidade de um sistema de controle de versões para modelos tridimensionais na sua funcionalidade de reversibilidade, e representa uma melhoria fundamental para estes sistemas.

1.3 Objetivos

Para responder a questão da seção anterior, é preciso identificar os objetivos que espera-se conseguir com esta tese, visto que um meio para resolver tal problema pode jamais existir ou não ser o ideal. Identificados como foco da tese, os sistemas de controle de versões para modelos tridimensionas e a operação de desfazer, pode-se estabelecer os seguintes objetivos específicos:

a) Localizar mudanças respeitando um determinado locus (lugar) do modelo;
b) Desfazer algumas mudanças de acordo com o locus preservando outras que não o afetam;
c) Gerar tutoriais a partir do locus previamente determinado; e
d) Gerar versões novas dos modelos com mudanças do locus desfeitas e preservando as que não o afetam.

1.4 Estratégia

Para realizar o Objetivo 1, adota-se o mapeamento de elementos de uma versão do modelo para outra de maneira a identificar os conjuntos de elementos que mudaram. Quando uma função bijetora entre os elementos não é possível, adota-se uma função injetora para realizar o mapeamento que é sempre feito da versão mais antiga (domínio) para a versão mais nova (contra-domínio). Se todos os elementos de uma versão do modelo possuem relações com uma versão anterior, é possível transferir as mudanças realizadas em uma versão à outra atualizando-se os elementos usados por transição entre conjuntos.

Para realizar o Objetivo 2, adota-se o paradigma da Divisão e Conquista para gerar subárvores de versão de acordo com o locus de interesse (Divisão), aonde é aplicado o desfazer, e em seguida tais mudanças provocadas pela técnica são agregadas à árvore principal (Conquista).

Para realizar o Objetivo 3, utiliza-se dos resultados obtidos do Objetivo 1, para determinar o locus de interesse específico e acompanhá-lo de acordo com seu mapeamento ao longo de sua construção produzindo um filme, ou uma sequência de quadros para cada mudança.

Para realizar o Objetivo 4, utiliza-se dos resultados obtidos dos Objetivos 1 e 2, para gerar novas versões a partir das modificações realizadas na árvore de versões através de sua execução no modelador.

1.5 Contribuição

As duas principais contribuições fornecidas por esta tese são:

a) O desenvolvimento de uma técnica para geração de subárvores (ou subconjuntos) especializadas para loci quaisquer de modelos tridimensionais. Isto permite a construção de tutoriais como produto direto do algoritmo implementado, i.e., a obtenção de caminhos da subárvore que exibem como regiões específicas foram geradas; e
b) A implementação do Desfazer Localizado, que é um desfazer especialista, para árvores de controle de versão (ou outras estruturas) em sistemas de controle de versões, que permite desfazer mudanças no modelo de acordo com o locus desejado enquanto busca-se preservar as regiões que não afetam tal locus. Até o presente momento desconhece-se qualquer trabalho similar existente com esta funcionalidade específica de Desfazer Localizado.

1.6 Organização

O restante desta tese se organiza nos capítulos que se seguem, onde se busca detalhar ao máximo todo o processo desenvolvido ao longo da pesquisa.

No Capítulo 2, é apresentada um breve histórico dos sistemas de controle de versões (Seção 2.1), já que a técnica depende de tais sistemas; em seguida é apresentada uma série de trabalhos que se baseiam em sistemas de controle de versões separados por classe, de modo a identificar a qual grupo esta tese pertence (Seção 2.2); em sequência este trabalho é divido em partes e os métodos relacionados são discutidos (Seção 2.3): mapeamento de geometrias (Subseção 2.3.1), rastreamento (*tracking*) de geometrias (Subseção 2.3.2), geração de tutoriais (Subseção 2.3.3) e desfazer (Subseção 2.3.4). Essas partes se alinham com a estratégia proposta no capítulo anterior.

No Capítulo 3, o novo método desenvolvido é descrito com a apresentação dos conceitos necessários para sua compreensão (Seção 3.1), dos pré-requisitos para a sua implementação em um sistema de controle de versões (Seção 3.2), de um resumo do algoritmo (Seção 3.3), das etapas para a realização do método (Seções 3.4, 3.5, 3.7, 3.8 e 3.9), do algoritmo utilizado em pseudo-código (Seção 3.10), e, por fim, de sua otimização implementada (Seção 3.11).

No Capítulo 4, os resultados obtidos para o método são apresentados de maneira que detalhes de implementação (Seção 4.1) e exemplos são fornecidos (Seções 4.2 e 4.3). Neste capítulo também é feita uma análise do método segundo critérios conhecidos (Seções 4.4 e 4.5),

e uma comparação com o estado da arte atual (Seção 4.6), para, então, discutir-se suas limitações atuais (Seção 4.7).

No Capítulo 5, o último capítulo desta tese, são apresentadas as suas implicações para a ciência em seu estado atual bem como as propostas de trabalhos futuros para pesquisadores interessados em perseguir este caminho (Seção 5.1).

2 TRABALHOS RELACIONADOS

Este capítulo discute os trabalhos relacionados. Ele encontra-se dividido em histórico dos sistemas de controle de versões (Seção 2.1), que apresenta um pouco da evolução desses sistemas sobre os quais a técnica depende, as categorias de métodos existentes para esses sistemas (Seção 2.2), e por fim, a descrição dos métodos existentes na mesma área e que se relacionam a alguma das partes da metodologia desenvolvida nesta tese (Seção 2.3).

2.1 Histórico dos sistemas de controle de versões

Um sistema de controle de versões (SCV) é uma ferramenta que armazena as mudanças feitas em um conjunto de dados e que possui três propriedades fundamentais: (a) anotações, para descrever e identificar a mudança realizada; (b) concorrência, para permitir que diversas pessoas atuem sobre um mesmo conjunto de dados; e, (c) reversibilidade, para reverter mudanças armazenadas (RAYMOND, 1992). O nome versão advém do fato de cada pessoa poder ter uma cópia dos dados com mudanças diferentes efetuadas sobre os mesmos. Se é possível mesclar estes conjuntos de mudanças quaisquer de duas ou mais pessoas de forma automática, isto é um problema Não Polinomial (NP) completo (PAPADIMITRIOU, 1997), que alguns SCVs tentam resolver com um grau de sucesso parcial.

Existem duas maneiras básicas para armazenar arquivos em um sistema de controle de versões: *Snapshots* e *Changesets* (POOL, 2004). *Snapshots* são obtidos quando um conjunto de dados completo é salvo cada vez que mudanças são efetuadas, o que, é claro, pode ocasionar problemas de armazenamento. *Changesets* são obtidos quando apenas as mudanças efetuadas, os chamados deltas, são salvas, e isso poupa armazenamento, mas custa processamento, já que um dado precisa ser reconstruído antes de ser exibido.

Um dos primeiros sistemas desse tipo a ser escrito foi o *Source Code Control System* (SCCS), desenvolvido por Marc Rochkind em 1972 (ROCHKIND, 1975). Quando ele foi executado pela primeira vez em um computador da IBM System/370 com o sistema operacional OS/MVT, marcou o início da primeira geração de SCVs como o principal sistema usado em máquinas Unix. O SCCS usava deltas intercalados para armazenar mudanças, contudo era usado principalmente para arquivos de texto (RUPARELIA, 2010).

Nesta primeira geração, os sistemas eram todos centralizados, de modo que todos os usuários acessavam um único conjunto de dados armazenados em um servidor, e a maioria

eram orientados ao acesso de arquivos, que é similar à metodologia de *snapshots*, onde todas as mudanças são salvas em um arquivo. Esses sistemas usavam tecnologia de trava (*lock*) para acesso de suas informações, i.e., somente um usuário por vez poderia efetuar mudanças em um arquivo. O acesso remoto a esses sistemas ainda era impossível, sendo uma característica que se tornou popular apenas com o advento da Internet anos mais tarde (RAYMOND, 1992).

A segunda geração tem início com a introdução do conceito de versão pelo SCCS, associando a cada mudança nos dados um id único, cujo formato ainda é usado em sistemas modernos. Um dos primeiros sistemas a aparecer após o SCCS foi o *Revision Control System* (RCS), escrito por Walter F. Tichy na universidade de Purdue no início dos anos 80 (TICHY, 1982b) que introduziu diversas funcionalidades para mesclagem de revisões (RUPARELIA, 2010). Mesclagem é o nome do processo de unir versões diferentes em uma única, unindo conjuntos de mudanças. O RCS é um dos sistemas mais antigos ainda ativo e presente nas distribuições atuais de sistemas operacionais baseados em Linux e Unix, fazendo parte do projeto GNU (*Gnu is Not Unix*) (TICHY, 1982a).

Nessa geração, o SVK[1] é implementado em cima do SVN em Perl para permitir um controle descentralizado limitado dos projetos (os sistemas existentes até então eram todos centralizados), além de uma operação de mesclagem mais robusta. A limitação na descentralização ocorre devido ao núcleo compartilhado com o SVN (APACHE SOFTWARE FOUNDATION, 2001).

A segunda geração dos sistemas de controle de versões também marca o aparecimento dos sistemas colaborativos remotos (o RCS possui tal funcionalidade). O *Concurrent Version System* (CVS) aparece no final da década de 80, e implementa a arquitetura de cliente e servidor. Contudo, ele é usado apenas de modo local (servidor e cliente na mesma máquina). A arquitetura TCP/IP (*Transmission Control Protocol/Internet Protocol* (SOCOLOFSKY; KALE, 1991) foi implementada nos sistemas apenas durante a década de 90. Após 10 anos de desenvolvimento estagnado nos sistemas de controle de versões, o *apache SubVersioN* 1.0 (SVN), em 2004 (ZEISS, 2004), introduz mesclagem (*merge*) para trabalhos colaborativos e o conceito de *Changesets* iniciando a terceira geração.

Os sistemas de controle de versões da terceira geração já são naturalmente descentralizados ou distribuídos, como o Git de Linus Torvalds (TORVALDS, 2011) e o Mercurial (MACKALL, 2005). Nesta geração, também surgem os sistemas baseados em *Changesets*

[1] Não é uma sigla.

(Conjuntos de Mudanças). Tom Lord, o autor de Arch (LORD, 2005), foi o responsável pela popularização do termo (RAYMOND, 1992). É importante notar, contudo, que esse conceito pode variar de sistema para sistema, e, portanto, pode armazenar informações diferentes para determinar uma mudança.

Esses sistemas ao longo de todas as gerações serviram para gerir arquivos de texto (BA *et al.*, 2013) ou códigos-fontes de programas (NEGARA *et al.*, 2012). No princípio o armazenamento era feito através de árvores de versão, e, apenas a partir da terceira geração, se começou a adotar *Directed Acyclic Graph*/grafo direcionado acíclico (*DAG*)s.

Hoje em dia, existem SCVs baseados em árvores os quais adotam o modelo de mesclagem antes de submissão (*merge* antes de *commit*), i.e., o usuário precisa resolver conflitos antes de poder gravar na árvore, e os SCVs baseados em *DAG*, que adotam o modelo de submissão antes de mesclagem (*commit* antes de *merge*) em que o usuário pode salvar suas modificações à vontade em um ramo do grafo antes de mesclar com o processo principal. Esses dois tipos são os mais usados, mas ainda há os que se baseiam em trava (*lock*), em que só um usuário pode trabalhar nos dados por vez; é possível também que um SCV permita mais de um tipo de opção para resolução de conflitos (WIKIPÉDIA, 2016; RAYMOND, 1992).

Recentemente, com a evolução dos SCVs, está surgindo uma quarta geração de SCVs especialistas que se utilizam de propriedades específicas das imagens para armazená-las (BAVOIL *et al.*, 2005; GRABLER *et al.*, 2009; CHEN *et al.*, 2011; CHEN *et al.*, 2016; HU *et al.*, 2013) ou que se utilizam de propriedades geométricas e topológicas de modelos tridimensionais para armazená-los (DENNING *et al.*, 2011; DOBOŠ; STEED, 2012a; DOBOŠ; STEED, 2012b; DENNING; PELLACINI, 2013; N.Y.U., 2005) [2]. Todos os sistemas de quarta geração herdam diversas características de seus antecessores, mas apresentam características novas e pertinentes aos tipos de dados para os quais são especializados.

Também recentemente, um novo tipo de sistema de versão simples surgiu baseado nos de primeira geração, ou estendendo as capacidades dos SCVs mais novos. Esse tipo particular de SCV busca atender às necessidades de artistas, e permite a exibição visual de mudanças, bem como desfazê-las; contudo, tais sistemas ou ainda não incorporaram todos os recursos já mencionados de SCVs tradicionais, ou não se utilizam das propriedades específicas de imagens e modelos tridimensionais para produção de novas funcionalidades (SKALNIK, 2013; FOLIO, 2016; PIXELAPSE, 2016; ALIENBRAIN, 2016; GITHUB, 2016). Alguns desses

[2] Doboš é um dos principais pesquisadores na área de SCVs para modelos 3D com o sistema 3D repos (DOBOŠ, 2016).

sistemas buscam agrupar múltiplas funcionalidades de diversos sistemas e de interesses do artista funcionam como canivetes suíços.

2.2 Categorias de métodos para SCV 3D

Nesta seção, os trabalhos sobre SCVs para modelos tridimensionais são classificados de acordo com sua área de conhecimento para determinar sobre qual tipo esta tese atua. Como o foco desta tese está em modelos tridimensionais, classifica-se a seguir as técnicas disponíveis para SCVs de quarta geração que atuam sobre estes modelos em três categorias de acordo com seu ramo do conhecimento na área de Banco de Dados (Subseção 2.2.1), de Visualização Computacional (Subseção 2.2.2), e de Processamento Geométrico (Subseção 2.2.3).

2.2.1 Banco de Dados

Técnicas nesta área buscam resolver questões que lidam com a implementação de um SCV para modelos tridimensionais, ou com a adição de recursos específicos pertinentes a este tipo de dados, desenvolvendo novos tipos de arquivos, metodologias de acesso, e aperfeiçoando o manuseio de informações geométricas para modelos tridimensionais entre outras funcionalidades.

No trabalho de Chou e Kim (1986) é desenvolvido um meio para implementar um sistema de controle de versões para um programa de *CAD*, abordando os aspectos operacionais e semânticos de um SCV: manipulação de mudanças, nomenclatura de versões, referências, notificações de mudanças, etc. Já o trabalho de Nyamsuren *et al.* (2013) apresenta uma plataforma (*framework*) para controle de versões na Web de modelos de *CAD*.

Doboš e Steed (2012b) também desenvolvem um banco de dados (Bando de Dados (BD)) para modelo 3D usando NoSQL (STROZZI, 1998). Eles abordam a questão de como armazenar grafos de cenas e desenvolvem um sistema de controle de versões baseado em submissão antes de mesclagem com *DAG*s. Cenas são representadas como coleções de modelos polimórficos juntos de seu histórico. Ao contrário de trabalhos anteriores, eles não se prendem a um tipo de modelador.

Mais tarde, Doboš *et al.* (2013) desenvolvem um novo formato de arquivo chamado XML3DRepo, que mistura o formato XML3D (DFKI *et al.*, 2011), que é uma extensão de código aberto para HTML com suporte para gráficos interativos 3D com navegadores através do WebGL (KHRONOS, 2014), e o formato 3D Repo (DOBOŠ, 2016), que é o resultado de seu trabalho

anterior e provê um SCV para modelos tridimensionais com acesso à um banco de dados NoSQL. Esse novo formato tenta determinar um meio eficiente para exibição e manipulação de modelos 3D na Web com o mínimo de requisições possível para transmissão de dados.

Há também outros trabalhos que focam exclusivamente nos aspectos de armazenamento e exibição de modelos geométricos que valem ser mencionados. O trabalho de ElYamany *et al.* (2012), por exemplo, busca aumentar a velocidade de acesso para modelos do tipo BIM (*Building Information Model*) através da construção de um *framework* especialista, já o trabalho de Brutzman e Daly (2004), que busca o desenvolvimento de uma tecnologia eficiente para exibição de dados tridimensionais na Web.

O *Vistrails Provenance Explorer for Maya* (N.Y.U., 2009) é um plugin intermediário entre um banco de dados e um SCV para geração de árvores do Vistrails (N.Y.U., 2005) no Maya (AUTODESK, 2016a). O plugin (uma extensão dos recursos do Maya) implementa as características de reversibilidade e anotação, porém sem a funcionalidade de concorrência. É sobre o produto deste plugin que a técnica desenvolvida nesta tese atua, embora seja possível utilizar outros SCVs ou BDs como base para sua implementação.

O trabalho desenvolvido nesta tese, portanto, não objetiva a construção de um BD ou de um SCV especialista. Ele apenas utiliza características dos SCVs como fundamentos de sua implementação, logo, qualquer SCV pode ser usado para implementar a técnica.

2.2.2 Visualização Computacional

Esta área analisa os dados armazenados ou gerados a partir de sistemas de controle de versões para visualização ou manipulação. Estes métodos buscam determinar áreas importantes do processo de construção de um modelo, assim como exibir os processos de construção da forma mais inteligível possível.

O trabalho de Su *et al.* (2009), embora não utilize modelos tridimensionais, e sim bidimensionais, adiciona a capacidade de visualização para um histórico de comandos em um editor gráfico vetorial. O interessante neste trabalho é que ele apresenta uma proposta de desfazer não-sequencial no histórico para modelos bidimensionais cuja idéia é similar à desta tese para modelos tridimensionais. O trabalho também referencia outros similares para tipos de objetos que não são modelos tridimensionais.

Seguindo na mesma linha, mas para modelos tridimensionais, o trabalho de Denning *et al.* (2011) acessa um sistema de controle de versões e adiciona visualização aos dados

tridimensionais para elaboração de um tutorial. Para diminuir o tamanho do tutorial e facilitar a compreensão do processo de construção de um modelo, as operações efetuadas são agrupadas por frequência e tipo, segundo um sistema de expressões regulares, e permite a adição de anotações a cada uma das etapas de construção.

Já o trabalho desenvolvido por Chen *et al.* (2014) busca localizar o melhor posicionamento da câmera para acompanhar a construção de um modelo. O algoritmo implementado analisa um histórico de comandos, e infere regiões importantes do modelo para que os pontos de visualização sejam gerados automaticamente através do posicionamento adequado de uma câmera e para que se busque visualizar o máximo de edições.

O sistema implementado por Freire *et al.* (2014) não é um SCV, mas objetiva a reprodução de experimentos científicos, e isto pode ser usado como um histórico visual (visualização) de como um modelo tridimensional foi construído por um artista usando arquivos XML e permitindo que anotações sejam inseridas nos passos realizados. O Vistrails Provenance Explorer Plugin (N.Y.U., 2009) se baseia nesse trabalho, permitindo que o sistema atue em conjunto com o modelador Maya (AUTODESK, 2016a) para gerar árvores compreensíveis ao Vistrails.

Por fim, o trabalho de Doboš *et al.* (2014) adiciona recursos de visualização sem um histórico de construção dos modelos tridimensionais. Cada arquivo do modelo 3D é tratado como um quadro-chave da construção de modo a produzir um histórico visual de edições. Esse trabalho usa de uma linha do tempo para exibir os quadros produzidos, e agrupa-os por tempo para facilitar a visualização. Tal trabalho parte da premissa que o modelo tridimensional foi feito salvando-se cada passo da construção, o que nem sempre é verdade.

Esta tese também não possui o foco na visualização de modelos tridimensionais, e pode ser usada em conjunto com os trabalhos descritos anteriormente, caso busque-se aumentar a qualidade dos produtos resultantes.

2.2.3 Processamento Geométrico

Esta área estuda os dados geométricos e topológicos de modelos tridimensionais armazenados ou gerados a partir de sistemas de controle de versões para a implementação de novas funcionalidades ou funcionalidades de SCVs especialistas.

Uma das funcionalidades mais pesquisadas atualmente é a de mesclagem de modelos tridimensionais, que contribui para a capacidade de concorrência de SCVs especialistas. O

trabalho de Doboš e Steed (2012a) desenvolve um algoritmo de mesclagem usando um grafo de cena. Para tal, um algoritmo de segmentação é usado dividindo o modelo em regiões, que são representadas como nós do grafo. Para comparar dois modelos, os grafos são verificados, e uma mesclagem dos elementos é feita por sobreposição de componentes usando um sistema de molas por proximidade para garantir contato (JAIN *et al.*, 2012), mas não necessariamente continuidade do modelo. Este trabalho em conjunto com os anteriores foi desenvolvido e compilado na tese de Dobŏs (DOBOŠ, 2015) em que um SCV especialista completo é descrito com suas funcionalidades fundamentais.

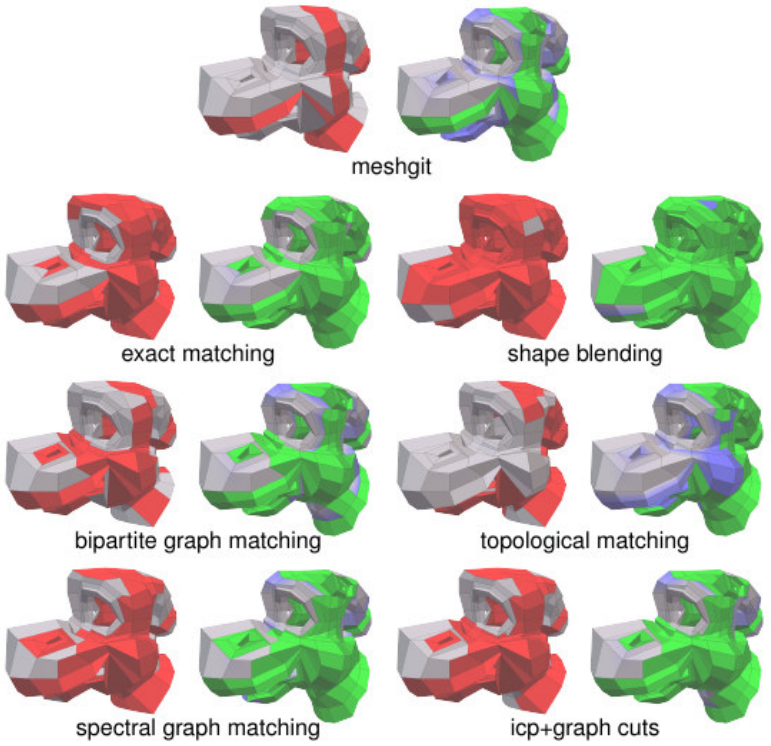
Em Denning e Pellacini (2013) foi desenvolvido um outro algoritmo de mesclagem para sistemas de controle de versões usando a distância de edição das malhas. Através de uma análise dos elementos geométricos e topológicos, criou-se uma fórmula arbitrária para aproximar modelos semelhantes, que possuam muitas correspondências, para tentar determinar modificações na malha (uma heurística para o cálculo de Conjuntos de Mudanças). Essa fórmula, ao contrário da definição apresentada nesta tese, produz um número grande de falsas referências como pode-se observar na Figura 2.1. Este trabalho em conjunto com os anteriores foi desenvolvido e compilado na tese de Denning (DENNING, 2014) em que um sistema para exibição e análise parcial de construção de modelos 3D é desenvolvido.

A técnica desenvolvida por esta tese, portanto, visa desenvolver uma nova funcionalidade para SCV especialistas focados em modelos tridimensionais. A técnica foca nesta área de processamento geométrico, e até o presente momento desconhece-se qualquer trabalho similar existente com esta funcionalidade específica.

2.3 Métodos

Nesta seção são apresentados métodos desta área proximamente relacionados com a técnica desenvolvida nesta tese. A técnica desenvolvida nesta tese é uma coleção de métodos que foram unidos para a produção de uma nova funcionalidade para SCVs de modelos tridimensionais (SCVs 3D). Nesta seção, descrevem-se as técnicas similares em mapeamento de geometrias (Subseção 2.3.1), rastreamento (*tracking*) de geometrias (Subseção 2.3.2), geração de tutoriais (Subseção 2.3.3) e desfazer (Subseção 2.3.4). Essas partes se alinham com a estratégia proposta do capítulo anterior (Seção 1.4).

Figura 2.1 – Em Denning e Pellacini (2013), é possível observar o uso de uma fórmula arbitrária baseada em distâncias para tentar determinar o *Changeset*, que embora produza resultados melhores que outras técnicas que não possuem esse fim, ainda assim produz muitas correspondências errôneas em vermelho.



Fonte: Denning *et al.* (2011).
Nota: Em verde e azul são demarcadas diferenças geométricas e topológicas.

2.3.1 Mapeamento de geometrias

Para realizar o mapeamento de uma versão do modelo a outra, foi usada a técnica de *hashing* geométrico, em que as coordenadas de um vértice são utilizadas para gerar uma chave única armazenada em tabelas de dispersão (*hash*). Similares a esta medida, no trabalho de Wolfson e Rigoutsos (1997), uma base vetorial é usada para gerar uma ~chave.

O *hashing* geométrico encontra sua principal aplicação em visão computacional para reconhecimento de características (*features*). É possível identificar um objeto a partir de fotos do mesmo tiradas de posições diferentes, e baseado em curvas e características particulares é possível inclusive computar transformações 3D e identificar objetos de uma maneira rápida e precisa com a técnica para aplicações médicas (GUÉZIEC *et al.*, 1997).

Outra aplicação do *hashing* geométrico é para reconhecimento de modelos 3D parcialmente oclusos em um cena (MIAN *et al.*, 2006). Visões distintas de um objeto são comparadas através de uma chave geométrica com um banco de dados dos modelos existentes,

Figura 2.2 – As tabelas de dispersão (*hash*) podem ser usadas para reconhecimento de modelos tridimensionais parcialmente oclusos (MIAN *et al.*, 2006).

Fonte: Mian *et al.* (2006).

e os modelos que encontram mais correspondências na tabela são transformados na cena para verificar qual deles se iguala ao modelo original (Figura 2.2).

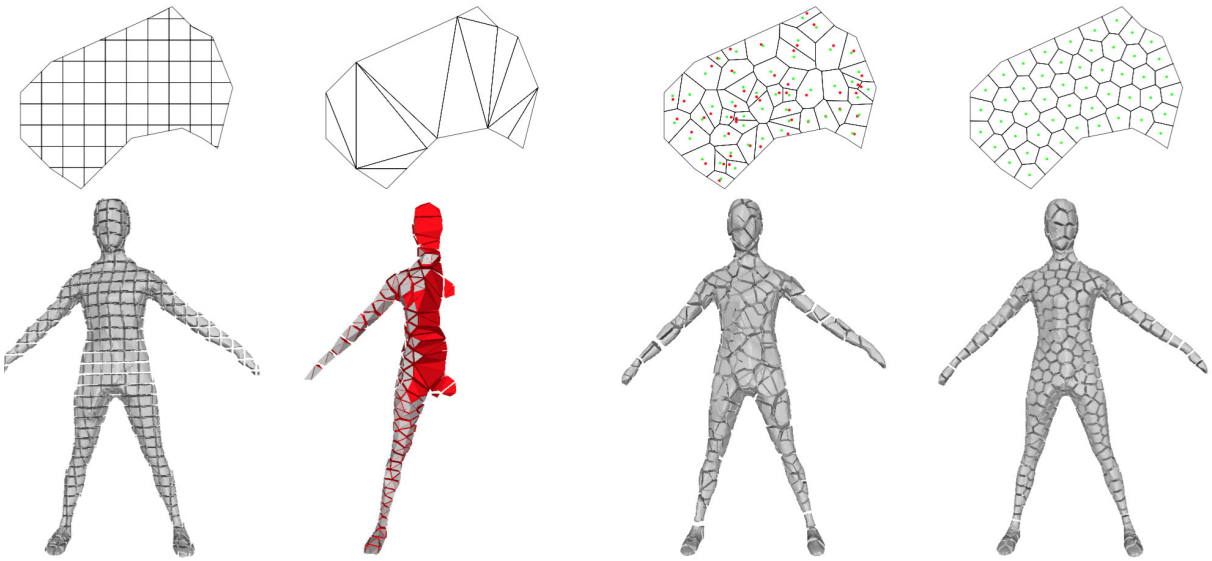
Encontrar similaridades entre modelos tridimensionais é um problema fundamental também em processamento geométrico, e existem tentativas de encontrar uma distribuição de probabilidades a partir de amostras aleatórias para efetuar esta correspondência (OSADA *et al.*, 2001). Também já implementou-se uma função de *hash* geométrico perfeita, injetora (LEFEBVRE; HOPPE, 2006), mas tal função exige que o modelo não sofra mudanças, o que não é adequado ao processo descrito nesta tese em que o modelo muda constantemente.

Nesta tese, parte-se da premissa que o modelo tridimensional é sempre o mesmo (i.e., que o usuário ao fazer o modelo de uma estrela não desiste, exclui o modelo atual, e começa a fazer o modelo de um rinoceronte); logo, não há necessidade de usar um banco de dados de modelos diversos ou características aleatórias para reconhecer similaridades entre modelos, apenas encontrar uma chave única que identifique os elementos do modelo para reconhecer elementos que não se alteram. Este é o primeiro desafio que a técnica implementada resolve de forma satisfatória.

2.3.2 Rastreamento de geometrias

O segundo desafio que esta tese enfrenta é de rastrear uma região de interesse do usuário, enquanto esta sofre deformações e mudanças de corpo rígido. Rastrear (*tracking*) um modelo tridimensional ou a região de um modelo não-rígido é uma tarefa difícil. O trabalho de Cagniart *et al.* (2010) é um dos primeiros a tentar solucionar esse problema através do uso de segmentos (*patches*) do modelo (dividindo o modelo em regiões). Para isto, os segmentos são considerados interdependentes de acordo com uma possível posição do modelo, mas a técnica

Figura 2.3 – Divisão espacial 3D de um modelo tridimensional para rastreamento durante uma animação.



Fonte: Du *et al.* (1999).

Nota: Da esquerda para direita observa-se, divisão por uma grade regular de voxels, tetraedralização por Delaunay, celúlas Voronoi com centros randômicos e células de Voronoi por centroídes.

não é capaz de lidar com pequenas mudanças (menores que os segmentos) e considera que a topologia do modelo permanece sempre constante.

No trabalho de Allain *et al.* (2015), pode-se observar a proposta de usar volumes para rastrear mudanças em modelos tridimensionais, o que é conseguido com sucesso em comparação aos outros métodos de segmentação, o que infere que volume é melhor para rastrear regiões tridimensionais do que superfícies ou segmentos (*patches*). No método implementado, Tesselação Central de Voronoi (DU *et al.*, 1999) é usada para encontrar os volumes (Figura 2.3). Contudo, mesmo neste caso, o volume falha em rastrear pequenas mudanças dentro dos volumes criados.

Trabalhos mais recentes (DEGTYAREV *et al.*, 2016) mantém a assertiva anterior e usam volumes para rastrear modelos tridimensionais durante uma captura de vídeo em tempo real. Essa técnica considera inclusive mudanças topológicas que o modelo possa sofrer ao contrário das técnicas anteriores. Infelizmente, essa técnica depende dos dados fornecidos por uma câmera que capture profundidade (*Red Green Blue Depth*/informações de câmera: vermelho, verde, azul e profundidade (*RGBD*); padrão de cor *Red Green Blue*/padrão de cores: vermelho, verde e azul (*RGB*), vermelho, verde e azul, mais profundidade), o que não se aplica à técnica desta tese atualmente. Contudo, acredita-se que seja possível realizar uma adaptação para trabalhos futuros.

As técnicas anteriores são usadas para animação de modelos tridimensionais, mas possuem similaridades com esta tese, pois a construção de um modelo tridimensional pode ser vista como uma animação em que o modelo se deforma e muda constantemente. Como um filme que exibe a evolução de um óvulo fecundado para um bebê antes do parto, SCVs 3D gravam informações que traduzem-se como um filme que exibe a evolução de um poliedro ou polígono para um modelo tridimensional mais complexo.

As técnicas anteriores também reforçam a ideia de que usar volumes para rastrear regiões do modelo tridimensional é um método adequado, embora existam limitações. Por exemplo, a técnica aqui descrita falha quando o conjunto de mudanças é maior que o volume e não menor, o que obriga o usuário a refazer uma seleção.

2.3.3 Geração de tutoriais

Após realizar o mapeamento de elementos entre todas as versões de um modelo tridimensional, e o rastreio de uma região de interesse do usuário de acordo com o mapeamento, é possível produzir uma sequência de imagens no formato de vídeo-tutorial para ensinar um artista como determinada região foi feita pela técnica.

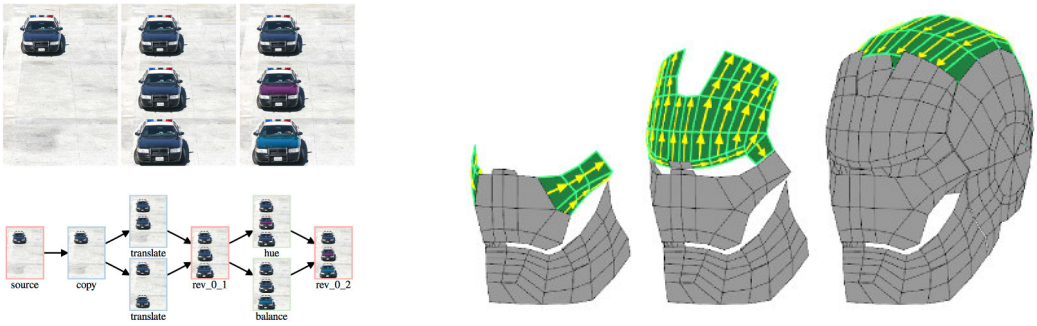
Nos últimos anos, o uso de vídeos na aprendizagem cresceu bastante com o advento da Internet, e websites como o YouTube. O uso de vídeos para aprendizado não é um meio novo, e faz parte de uma metodologia chamada Aprendizado Baseado em Demonstração (*Demonstration-Based Learning, Demonstration Based Learning/ensino baseado em demonstração (DBL)*; MA *et al.*, 2010). Texto como metodologia de ensino precisa ser lido e compreendido, o que consome tempo e energia mental. Já vídeos podem ser acelerados para reduzir o tempo de transmissão de informação, e podem exibir o objeto de estudo ao invés de fornecer uma descrição (PONZANELLI *et al.*, 2016) (imagine um texto que explica como andar de bicicleta, e um vídeo-tutorial da mesma atividade).

Recursos visuais gerados usando modelos tridimensionais ajudam no aprendizado tanto de crianças (JADEJA *et al.*, 2016) quanto de estudantes de medicina (PUJOL *et al.*, 2016). É razoável, então, inferir que artistas possam aprender através do trabalho de outros artistas com o auxílio de recursos visuais similares.

De fato, tutoriais similares produzidos por esta técnica, sem o uso de vídeos, já foram usados para o ensino de artistas tanto em modelagem, quanto em edição gráfica bidimensional conforme pode ser visto na Figura 2.4. Os vídeo-tutoriais produzidos por esta tese não recebem

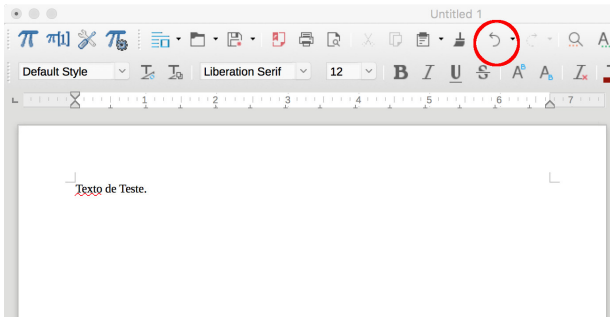
Figura 2.4 – Tutoriais produzidos usando sistemas de controle de versões: a) tutorial que demonstra o processo de aplicação de um filtro (CHEN *et al.*, 2011); b) tutorial que demonstra o processo de construção de uma malha (DENNING *et al.*, 2011).

(a) (b)



Fonte: (a) Chen *et al.* (2011); (b) Denning *et al.* (2011).

Figura 2.5 – Ferramentas de texto, Writer do Libreoffice (THE DOCUMENT FOUNDATION, 2016), que implementa a operação de desfazer.



Fonte: o autor.

Nota: Circulado em vermelho o botão padrão utilizado por ferramentas similares (uma seta em formato circular no sentido anti-horário). Esta operação também pode ser ativada em vários programas de diversos sistemas operacionais através do conjunto de teclas CTRL + Z ou ⌘ + Z.

anotações, porém a existência das mesmas não é necessária para que ocorra o aprendizado (NAKAMURA; IGARASHI, 2008).

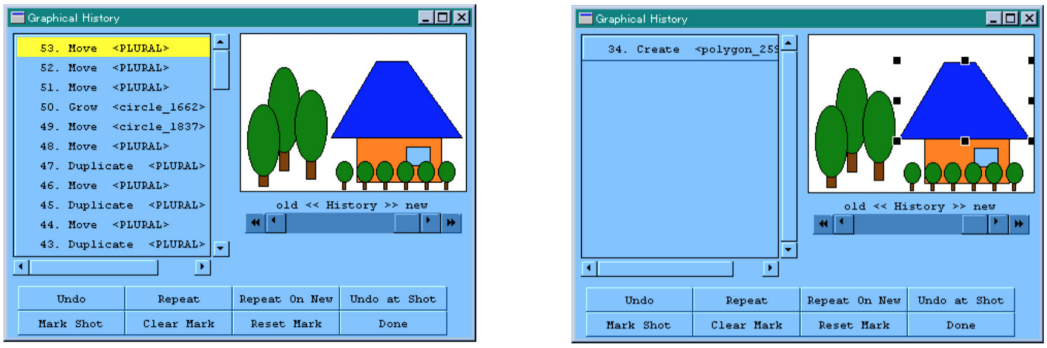
2.3.4 Desfazer

Desfazer é uma operação que permite retroceder uma ação efetuada sobre um programa qualquer, e só pode ser feita a partir do momento que alguma ação seja realizada sobre este programa (Figura 2.5). Essa operação, além de ser muito comum em vários programas como editores de texto, editores de imagem e modeladores 3D, também são encontradas em versões especialistas em diversos artigos científicos.

Um exemplo de trabalho que implementa um desfazer especialista para textos é o de Li e Li (2003) em que um Desfazer Localizado por regiões é implementado. Nessa técnica, uma parte do texto é selecionada e aplica-se sobre ela um desfazer parcial. Esse é um dos

Figura 2.6 – Desfazer Localizado baseado em regiões para imagens em que um histórico de operações foi realizado para produzir a figura em (a). Ao selecionar o teto da casa em (b) (pontos quadrados pretos dispostos em forma retangular), é possível desfazer apenas as operações que atuam sobre o teto da casa (para este caso, a operação 34).

(a) (b)



Fonte: Meng *et al.* (1998).

primeiros trabalhos conhecidos de desfazer especialista. Em seguida, este trabalho é aplicado sobre planilhas para permitir o desfazer parcial, selecionando-se operações no histórico que afetam determinada linha de uma planilha sem usar um SCV (KAWASAKI; IGARASHI, 2004).

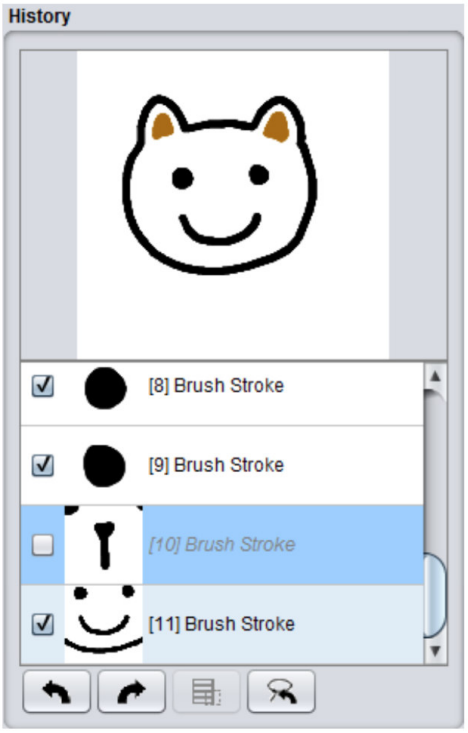
O Desfazer Localizado por regiões também já foi implementado para imagens, conforme visto no trabalho de Zhou e Imamiya (1997), permitindo a seleção global ou local para efetuar o desfazer. Essa operação também é realizada em Meng *et al.* (1998), como pode ser visto na Figura 2.6. O Desfazer Localizado também já foi idealizado para grandes dispositivos gráficos que permitam a interação entre dois ou mais usuários de forma colaborativa (SEIFRIED *et al.*, 2012).

O Desfazer Localizado para modelos tridimensionais jamais foi implementado até o presente momento (desconhece-se qualquer trabalho similar que implemente tal funcionalidade específica). Deste modo, esta tese é inédita ao apresentar essa nova funcionalidade para tais modelos. Além do alvo diferenciado, também se destaca dos outros trabalhos apresentados, pois imagens e textos possuem uma estrutura geométrica regular de representação por forma vetorial ou matricial que facilita tal operação.

Há um segundo tipo de desfazer especialista já implementado na literatura chamado de Desfazer Seletivo (*Selective Undo*). Essa operação pode ser confundida com o Desfazer Localizado, pois permite desfazer operações de forma não-linear no histórico, mas suas metodologias são diferentes.

A principal diferença entre um Desfazer Localizado e o Seletivo é que este último não seleciona regiões espaciais (2D ou 3D) conforme o primeiro, mas, sim, operações armazenadas

Figura 2.7 – Desfazer Seletivo para um programa de imagens: seleciona-se uma operação do histórico e remove-a do histórico desmarcando-se o símbolo de visto (✓) na caixa correspondente.



Fonte: Myers *et al.* (2015).

Nota: Na figura, pode-se observar que o usuário removeu do histórico de forma não-linear a operação que adicionava um focinho ao desenho de um gato. Note-se que não há seleção de regiões do desenho (MYERS *et al.*, 2015).

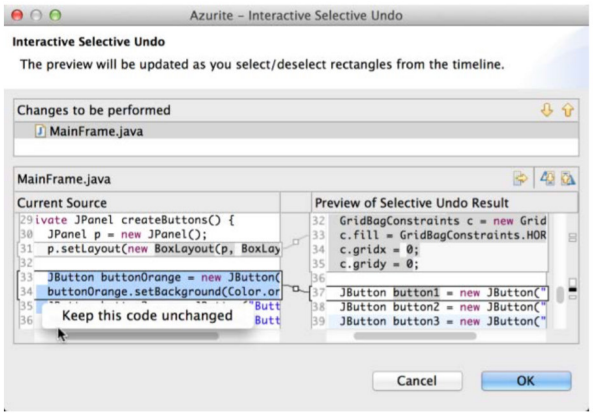
em um histórico de comandos. Deste modo, o desfazer seletivo busca remover operações dependentes a partir de informações armazenadas em um histórico de comandos (e.g., o identificador de sessão de um usuário) selecionando-se diretamente os comandos do histórico (e não regiões espaciais).

No trabalho de Su *et al.* (2009) existe um exemplo de Desfazer Seletivo para modelos gráficos bidimensionais. E o trabalho de Myers *et al.* (2015) também fornece um outro exemplo de Desfazer Seletivo para imagens, conforme pode ser visto na Figura 2.7, em que o focinho do desenho de um gato é removido.

O Desfazer Localizado já foi implementado parcialmente confundindo-se com um Desfazer Seletivo para editores de código (texto) conforme demonstrado no trabalho de Yoon e Myers (2015). Essa confusão é compreensível em textos, e somente para textos, visto que a noção de dimensão espacial pode não ser aplicada a um texto representado como sequência de caracteres.

Um código-fonte escrito é desfeito de forma não-linear sobre o sistema implementado

Figura 2.8 – No sistema AZURITE, há uma mistura dos conceitos de Desfazer Localizado e Seletivo.



Fonte: Yoon e Myers (2015).

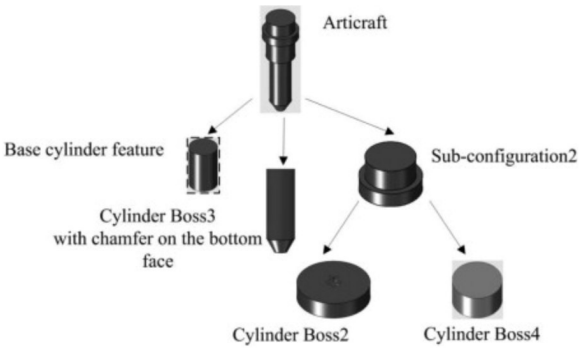
Nota: Após selecionar uma operação do histórico que afeta uma classe ou método específico é possível selecionar um retângulo de texto que deve ser preservado ao desfazer operações de forma iterativa (o usuário pode alterar o tamanho e quantidade de retângulos) (YOON; MYERS, 2015).

chamado AZURITE (YOON; MYERS, 2015). Primeiro, o usuário seleciona um ponto do histórico de comandos de onde todas as operações que afetam uma determinada classe ou método devem ser reunidas e, em seguida, interativamente seleciona-se, através de um retângulo, as operações que deseja-se manter enquanto as outras são desfeitas até retornar-se ao comando mais recente do histórico que afeta aquela classe ou método (Figura 2.8).

A técnica desenvolvida por Cheng *et al.* (2014) funciona para sistemas de modelagem *CAD*, e tira vantagem da maneira que tais sistemas de modelagem são organizados, i.e., por composição de modelos simples através de operações Booleanas, para ser capaz de efetuar o Desfazer Seletivo (essa composição também infere uma hierarquia de operações). A técnica também considera um cenário com vários usuários já que não usa um SCV 3D, mas difere-se do Desfazer Localizado, visto que não seleciona regiões e não funciona para um modelo tridimensional (composto por uma malha poligonal) obtido nos modeladores como Blender (BLENDER, 2016) e Maya (AUTODESK, 2016a). Esses modelos não são formados por operações que dependem de outras ou por composição de modelos simples. Na Figura 2.9 pode-se ver um exemplo complexo usado pelo trabalho mencionado.

Outro exemplo de Desfazer Seletivo, voltado para modeladores 3D convencionais levando em consideração trabalhos de vários usuários sem um SCV 3D, é o trabalho de Cheng *et al.* (2012). Nesse trabalho, busca-se desfazer uma operação encontrando-se todas as operações dependentes de uma dada operação. Contudo, sem um mecanismo de rastreio geométrico ou topológico, mesmo o Desfazer Seletivo só funciona para casos simples (a Figura 2.10 demonstra

Figura 2.9 – Este é um dos exemplos de modelos complexos sobre os quais a técnica de Desfazer Seletivo atuou em um modelador *CAD* (CHENG *et al.*, 2014).



Fonte: Cheng *et al.* (2014).

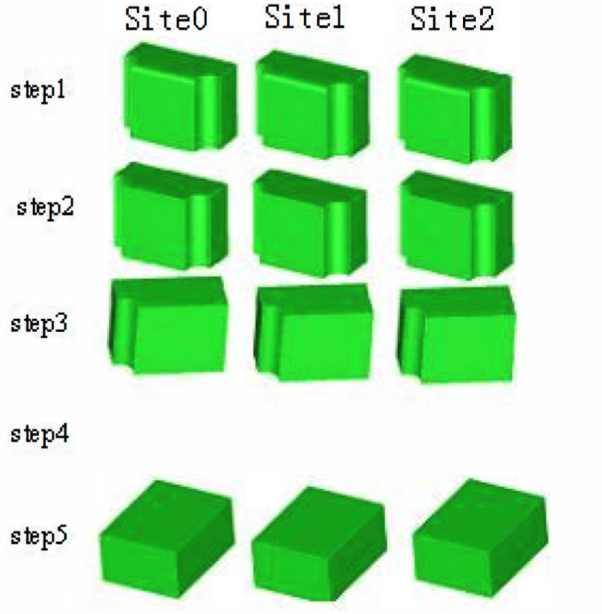
o modelo utilizado como resultado do Desfazer Seletivo por tal trabalho). Como já foi dito, ao contrário de programas de *CAD*, não há uma hierarquia natural entre os componentes de um modelo.

Um terceiro exemplo de desfazer especialista é um que atua sobre um sistema colaborativo de modelos *Constructive Solid Geometry*/geometria de construção de sólidos (*CSG*) (*Constructive Solid Geometry*) sem o uso de SCV 3D, conforme implementado por Cheng *et al.* (2009), chamado de Desfazer Multi-usuário. O efeito da operação de modo local e compartilhado é feito usando o seu id, e depende das relações hierárquicas existentes entre modelos *CSG*, similar ao que ocorre com modelos *CAD*. Esse tipo de desfazer especialista pode ser implementado de forma complementar aos outros tipos de operações de desfazer, caso não usem um SCV 3D.

Um último trabalho que merece menção, embora não implemente um desfazer especialista, é a dissertação de Nuss (2016) onde são discutidas as expectativas de um usuário com relação a uma ferramenta de desfazer em um ambiente colaborativo de *CAD* de acordo com um questionário realizado. A dissertação aborda questões como operações locais e globais de desfazer, e o compartilhamento dessas operações entre usuários.

No próximo capítulo (Capítulo 3), a técnica desenvolvida nesta tese é apresentada em detalhes.

Figura 2.10 – Este modelo é o único usado para realizar Desfazer Seletivo em modeladores 3D conforme o artigo de Cheng *et al.* (2012).



Fonte: Cheng *et al.* (2012).

3 MÉTODO

Nas próximas seções fornecem-se a base conceitual necessária (Seção 3.1), os pré-requisitos (Seção 3.2), e um resumo do método (Seção 3.3). Em seguida, detalha-se o funcionamento de cada passo do algoritmo da seguinte maneira: o cálculo do *Changeset* (Seção 3.4), a classificação das operações (Seção 3.5), a propagação da seleção (Seção 3.6), os testes de interseção utilizados (Seção 3.7), o processo de geração da subárvore (Seção 3.8), e a remoção de nós com o Desfazer Localizado (Seção 3.9). Ao final do capítulo, exibem-se o pseudo-algoritmo para geração de tutoriais e para o Desfazer Localizado (Seção 3.10), assim como o processo de otimização implementado para o algoritmo (Seção 3.11).

3.1 Conceitos

Esta seção descreve os conceitos necessários para compreensão do trabalho realizado. As subseções seguintes definem Vértices (Subseção 3.1.1), Arestas (Subseção 3.1.2), Vetor (Subseção 3.1.3), Face (Subseção 3.1.4), Normal (Subseção 3.1.5), Malha (Subseção 3.1.6), Região de uma malha (Subseção 3.1.7), Operação (Subseção 3.1.8), Nó (Subseção 3.1.9), Árvore de Controle de Versão (Subseção 3.1.10), Conjunto universo (Subseção 3.1.11), Sequência de Construção (Subseção 3.1.12), Versão (Subseção 3.1.13), Conjunto de Mudanças ou *Changeset* (Subseção 3.1.14), Desfazer (Subseção 3.1.15), Desfazer Localizado (Subseção 3.1.16), Tutorial (Subseção 3.1.17), Seleção (Subseção 3.1.18) e Caixa envoltória (Subseção 3.1.19).

3.1.1 Vértice

É uma posição do espaço linear $\mathbb{R}^3$ descrita por três coordenadas reais. Para esta tese, vértices são únicos, i.e., não podem se repetir.

3.1.2 Aresta

Dois vértices no $\mathbb{R}^3$ que se unem por uma linha formam uma aresta. Uma aresta não possui direção ou espessura. Arestas também são únicas.

3.1.3 Vetor

Dois vértices no $\mathbb{R}^3$ que se unem por uma linha direcionada (seta) formam um vetor no $\mathbb{R}^3$. Um dos vértices define a origem da linha, e o outro o seu destino. Vetores não são únicos, e arestas podem ser transformadas em vetores.

3.1.4 Face

É a união de pelo menos três arestas de modo que cada um dos vértices seja compartilhado entre duas arestas diferentes, e que cada aresta conecte-se a pelo menos duas arestas diferentes. Também chamada de polígono, que é uma sequência de várias linhas conexas cujo vértice inicial da primeira linha corresponde ao final da última linha, a menor face possível é um triângulo. Faces são únicas e determinam que os seus vértices devem ser percorridos no sentido horário para esta tese. Este trabalho só considera faces triangulares e quadrangulares.

3.1.5 Normal

É um vetor ortogonal à uma face caso esta esteja no espaço $\mathbb{R}^3$; já no espaço $\mathbb{R}^2$, é um vetor perpendicular a uma aresta. Para esta tese, todas as normais são unitárias e, portanto, únicas, i.e., não há mais de uma normal com mesma direção a partir de um mesmo ponto do espaço para cada malha.

3.1.6 Malha

É um conjunto de faces unidas de modo que toda aresta existente seja compartilhada entre duas faces diferentes, e que o conjunto de interseção entre quaisquer duas faces diferentes contenha uma única aresta ou seja vazio.

3.1.7 Região de uma malha

É um subconjunto das faces pertencentes a uma malha. Todo subconjunto de faces possui uma fronteira, isto é, uma sequência de arestas que ligam-se a apenas uma face.

3.1.8 *Operação*

É a transformação de uma malha no $\mathbb{R}^3$ através da adição, remoção, ou alteração de um subconjunto de elementos da malha (e.g., vértices, arestas, e faces). Exemplos de operações são: excluir uma face, rotacionar parte de uma malha, adicionar cores em uma malha qualquer, etc. A Figura 3.1 exibe a estrutura de uma operação. Quando uma malha sofre uma operação, a sua versão (Subseção 3.1.13) é incrementada em uma unidade.

3.1.9 *Nó*

É uma unidade de informação que contém ids, labels, relações entre nós, e funções *API* (*Application Program Interface*), ou uma operação (Subseção 3.1.8).

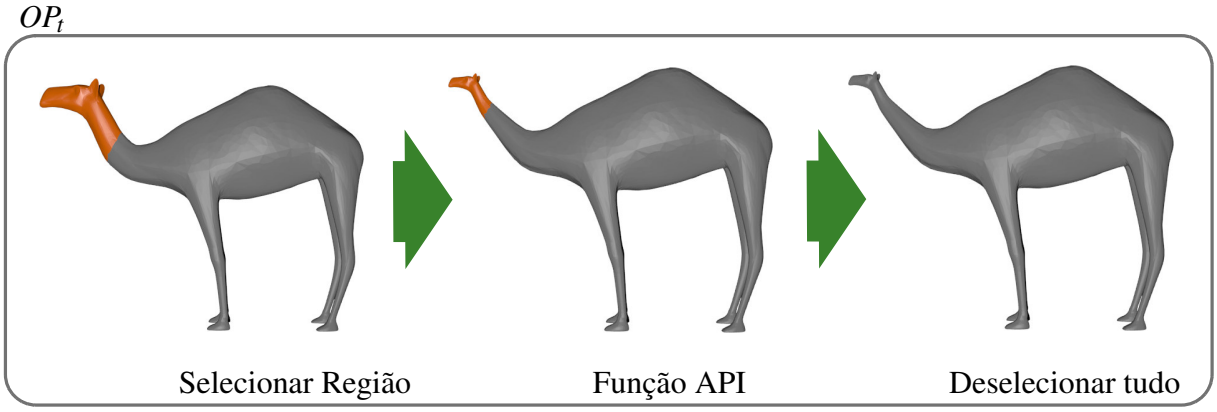
3.1.10 *Árvore de Controle de Versão (ACV)*

É uma estrutura de dados usada por sistemas de controle de versões; um conjunto de nós sem ciclos. Também pode ser chamada de árvore de versões.

3.1.11 *Conjunto Universo*

É o espaço vetorial formado pela combinação linear entre os 3 eixos canônicos: $\vec{i} = (1,0,0)$, $\vec{j} = (0,1,0)$, e $\vec{k} = (0,0,1)$. Todas as malhas correspondem ao resultado de uma operação armazenada em uma árvore de versão, e estão inseridas no Conjunto Universo.

Figura 3.1 – Uma operação (OP_t) é um nó de uma árvore de versões, e é formado pela seleção de uma região (em laranja), a função *API* com seus parâmetros, e a deseleção da região determinada previamente (Seção 3.2).



Fonte: o autor.

Nota: O conjunto de elementos selecionados pode ser vazio.

3.1.12 Sequência de Construção

É um caminho em uma árvore. Uma sequência de nós que se inicia na raiz (i.e., um nó sem pais) e termina em uma folha (i.e., um nó sem filhos).

3.1.13 Versão

É o estado de uma malha (i.e., a configuração dos elementos da malha: sua localização e propriedades) depois que todas as operações de uma Sequência de Construção em uma ACV foram aplicadas (POOL, 2004). Também corresponde a um atributo do vértice pertencente ao conjunto natural.

*3.1.14 Conjunto de Mudanças (***Changeset***)*

É o conjunto de diferenças entre duas malhas no $\mathbb{R}^3$ de modo que a diferença entre suas versões seja maior ou igual a 1. Todos os elementos da malha (e.g., vértices, arestas, e faces) modificados por uma operação fazem parte do *Changeset*, isto é, uma região da malha que foi afetada por uma única operação realizada na menor versão ou versão anterior.

3.1.15 Desfazer

É uma operação comum em modeladores 2D e 3D. Ela transforma uma malha para um estado prévio antes que uma ou mais operações fossem realizadas (sua versão anterior). O desfazer retorna em pelo menos uma unidade no atributo de versão.

3.1.16 Desfazer Localizado

Gera uma nova versão da malha em que o estado prévio para uma região da malha é restaurado com a possível remoção de alguma característica ou região da malha, i.e., somente as operações que afetam tal região ou característica são desfeitas. Essa nova operação é implementada pelo método exposto nesta tese, e não existe em modeladores 3D atuais.

3.1.17 Tutorial

É qualquer Sequência de Construção percorrida gradualmente da raiz à um nó em que cores são usadas para demonstrar o *Changeset* obtido a cada versão da malha até sua versão

final que corresponde ao nó final da sequência. Demonstra a produção de uma malha qualquer ou uma região da mesma.

3.1.18 Seleção

É uma região da malha obtida através de uma Caixa Envoltória (Subseção 3.1.19), i.e., um subconjunto da malha contendo vértices, arestas ou faces. Uma seleção também pode ser obtida de outras maneiras fornecidas por um modelador 3D, e tais métodos de seleção fazem parte das operações armazenadas em nós como parâmetros de funções *API* (Subseção 3.1.9); contudo quando o método exposto nesta tese efetua uma seleção trata-se sempre da região obtida pela Caixa Envoltória.

3.1.19 Caixa Envoltória

É uma região do espaço em formato de hexaedro que envolve um subconjunto da malha com vértices, arestas ou faces. A caixa envoltória determina uma seleção, possuindo seis faces, doze arestas, e oito vértices que determinam os planos delimitadores de uma região do Conjunto Universo no $\mathbb{R}^3$. É chamada na literatura científica de *Bounding Box* (CHANG *et al.*, 2011).

3.2 Pré-requisitos

Nessa seção os pré-requisitos para o algoritmo são discutidos. O método proposto requer como entrada uma árvore de versões (ou um Grafo Direcionado Acíclico) e uma seleção feita pelo usuário através de uma Caixa Envoltória (CHANG *et al.*, 2011). A árvore deve conter todos os comandos feitos pelo usuário e pertencentes a *API – Application Program Interface* – (CORBET *et al.*, 2005) do programa de modelagem utilizado para produzir uma malha. Para esta tese, o programa para modelagem utilizado foi o Maya (AUTODESK, 2016a), e as árvores foram geradas através do Vistrails Provenance Plugin para Maya (N.Y.U., 2009) que foi usado para gerenciar as versões das malhas como um SCV.

A raiz de uma árvore de versões é uma operação simples (e.g., a inclusão de um poliedro ou um polígono na cena), e todo nó existente na árvore corresponde a uma operação

feita na geração da malha poligonal, isto é, um comando efetuado pelo usuário enquanto usava o modelador. Os nós de uma árvore de versões devem conter também os parâmetros correspondentes a uma região da malha qualquer se existentes (seleção), uma função *API* de um modelador 3D, e a deseleção da região determinada previamente (Figura 3.1). Isto é necessário para garantir que o sistema de controle de versões possua a confiabilidade das informações após a aplicação do método (de acordo com os princípios do Atomicidade, Consistência, Isolamento e Durabilidade (ACID)) (GRAY, 1981) das operações realizadas. Destaca-se que um nó pode conter apenas uma função *API* quando a seleção é desnecessária ou vazia (e.g., uma operação para efetuar um zoom ou mover a câmera).

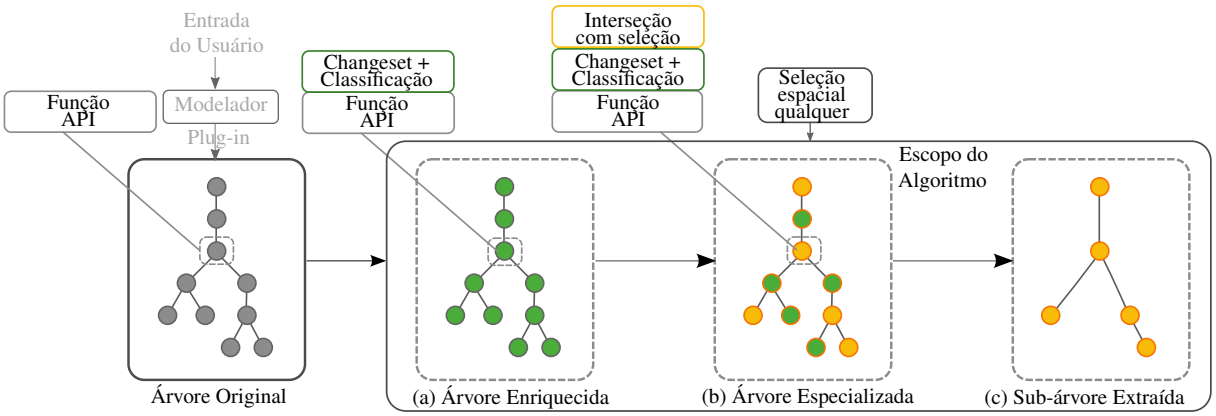
O sistema de controle de versões deve garantir a confiabilidade das transações efetuadas de modo que um nó possa ser removido da árvore de versão sem deixar o banco de dados em um estado de conflito (embora o Vistrails Provenance Explorer – N.Y.U. (2009) – não implemente tal recurso, ele foi implementado pelo autor para esta tese). Por exemplo, se duas operações sucessivas usam uma mesma seleção (os mesmos parâmetros), então o sistema que monitora a modelagem sendo feita pelo usuário e constrói uma árvore de versões deve: selecionar uma região, efetuar a primeira operação, deselecionar a região, e inserir um novo nó com estes dados na árvore (Figura 3.1), e, então repetir a mesma seleção, antes de iniciar a segunda operação, deselecionar novamente, e gravar o segundo nó. Esse processo garante o isolamento das transações de acordo com o ACID.

O método implementado extrai o *Changeset* (POOL, 2004) de cada operação, e o usa para efetuar mapeamentos entre as malhas no $\mathbb{R}^3$. Portanto, a seleção dentro de um nó deve conter os elementos (i.e., vértices, arestas e faces) nos quais a função *API* atua como já foi dito. Uma conexão entre os atributos dos elementos, tal qual o modelador Maya permite (AUTODESK, 2016a), também é considerada como uma seleção (parâmetros) para o propósito desta tese.

3.3 Resumo do algoritmo

Nesta seção um resumo do algoritmo é apresentado. O algoritmo recebe uma árvore de versões, e uma seleção determinada (que foi aplicada em uma versão qualquer) como entrada e procede da seguinte maneira (Figura 3.2): a) as correspondências entre vértices são calculadas

Figura 3.2 – Resumo do algoritmo (Seção 3.3) com a árvore original e uma seleção determinada em cinza como sua entrada: a) Em verde, todos os nós da árvore original são enriquecidos com a informação do seu *Changeset* e sua classificação; b) Cada nó com um *Changeset* que se intercepta com a seleção arbitrária é marcado em amarelo; c) Todos os nós não-marcados são removidos para se obter a subárvore final.



Fonte: o autor.

entre versões distantes de uma unidade, e os *Changesets* das operações são obtidos; b) os nós são classificados (Seção 3.5) como locais ou globais de acordo com o conjunto obtido anteriormente; c) a seleção da entrada é propagada ao longo da árvore de versões, primeiro na direção da raiz e em seguida das folhas, respeitando a classificação dos nós (Seção 3.6) – busca em profundidade (CORMEN *et al.*, 2001) foi usada para visitar todos os nós, ainda durante a propagação, a intersecção entre a seleção propagada e o *Changeset* para cada nó é computado; d) uma subárvore (i.e., um subconjunto da árvore) é criada usando os conjuntos de intersecções obtidos (Seção 3.8) como critério para a escolha de nós. A Figura 3.2 exibe um resumo gráfico do algoritmo. Para o caso do Desfazer Localizado, dois passos adicionais são realizados: a) a remoção de nós locais contidos na subárvore com a atualização dos parâmetros das funções *API* do modelador; e b) geração de novas versões para as Sequências de Construção que foram alteradas pela remoção de nós. Nas próximas seções, os passos do algoritmo serão detalhados com mais profundidade, começando pelo cálculo dos Changesets, Seção 3.4, que depende da correspondência entre vértices de versões vizinhas.

3.4 Cálculo dos Changesets

Para calcular os conjuntos de mudanças entre duas versões consecutivas e atualizar os parâmetros de funções *API*s após a remoção de nós, é preciso identificar como os vértices

de uma versão mapeiam-se para a versão seguinte (outros elementos da malha são mapeados usando os vértices como referência). Mapear para esta tese significa encontrar uma relação de correspondência entre os elementos de malhas distintas.

Para efetuar o mapeamento dos vértices, estabeleceu-se três tipos possíveis de casos:

a) Vértices que não se alteram entre versões diferentes;
b) Vértices que se alteram entre versões diferentes, e não mudam de quantidade; e
c) Vértices que se alteram entre versões diferentes, e mudam de quantidade.

Identificar os vértices que não se alteram entre versões diferentes já é o suficiente para capturar o Conjunto de Mudanças (*Changeset*) de uma malha qualquer. Contudo, para poder efetuar o Desfazer Localizado, é preciso atualizar os parâmetros de uma operação, o que corresponde aos dois itens seguintes. Explica-se nas subseções seguintes o tratamento de cada um dos casos: vértices que não se alteram (Subseção 3.4.1), vértices que se alteram, e não mudam de quantidade (Subseção 3.4.2), e vértices que se alteram, e mudam de quantidade (Subseção 3.4.3).

3.4.1 Vértices que não se alteram

Para encontrar os vértices que não se alteram, inicialmente faz-se uso de uma técnica conhecida como *Geometric Hashing* (WOLFSON; RIGOUTSOS, 1997). Essa técnica consegue agrupar características geométricas de modelos diferentes em *buckets* para comparação em um tempo polinomial linear, inclusive $\mathcal{O}(1)$.

No Algoritmo 1, em que t é a versão da malha M, usou-se as potências de 10007 (linha 2), um número primo, para criar um sistema numérico que fornece uma chave única para cada vértice existente de acordo com suas três primeiras coordenadas (linha 6). Através da posição da vírgula no número real de uma coordenada, estabelece-se uma aproximação de até quatro casas decimais (linhas 7 a 13; até 10 casas podem ser usadas, mas, neste caso, usa-se um número primo maior para evitar vértices com chaves repetidas, i.e., conflitos). Essa aproximação implica que todos os vértices de coordenadas similares em até quatro casas decimais possuem a mesma chave. Em caso de conflito, uma lista encadeada é mantida com os vértice de mesma chave.

Uma vez que se possui todas as chaves *hash* para uma malha M_t, repete-se o processo para a malha M_{t+1}. Desta maneira, todos os vértices semelhantes são adicionados aos *buckets* correspondentes. Este mapeamento de vértices que não se alteram é necessário, pois, sem ele, é

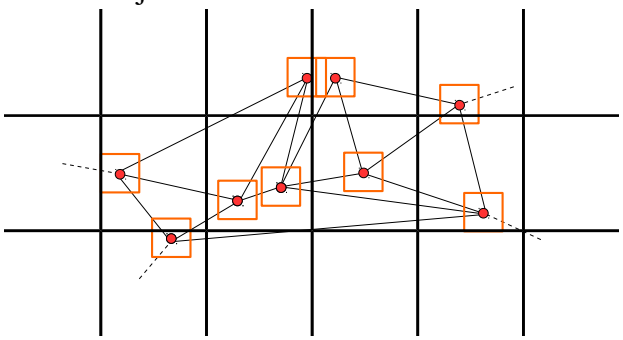
Algoritmo 1: Calcula uma chave_hash por meio de um sistema numérico de base 10007.

```
1  início
2      potencias ← {1.0f, 10007.0f, 100140049.0f};
3      chave_hash ← 0;
4      expoente_primo ← 2;
5      para cada vértice de M_t faça
6          para cada coordenada c de vértice faça
7              pos ← local. da vírgula em c;
8              se pos ⩽ 4 então
9                  expoente_decimal ← 4 - pos;
10             fim
11             senão
12                 expoente_decimal ← pos - 4;
13             fim
14             c = c * 10^expoente_decimal;
15             chave_hash ← chave_hash + c * potencias[expoente_primo];
16             expoente_primo ← expoente_primo - 1;
17         fim
18     fim
19     retorna chave_hash;
20 fim
```

necessário comparar cada vértice de uma malha M_t com todos os vértices da malha M_{t+1}.

Para aumentar ainda mais a velocidade dos testes, e reduzir o número de conflitos implementou-se ainda a divisão hierárquica do espaço através de uma grade constituída por hexaedros homogêneos que contém as chaves *hash* pertencentes aos vértices no lugar dos vértices, conforme pode ser visto na Figura 3.3. Uma divisão hierárquica do espaço é uma árvore de dados em que em sua raiz está um espaço coordenado real limitado, e seus filhos representam subdivisões deste espaço que podem ser homogêneas ou heterogêneas. Na octree (BRUNET; NAVAZO, 1990), esta divisão é feita de 8 em 8 regiões por nível da árvore, mas, nesta tese, implementa-se de forma que um novo nível da árvore, que pode ter mais de 8 filhos, só é inserido para reduzir o número de conflitos nas tabelas de dispersão.

Figura 3.3 – Um pedaço de uma malha com vértices em vermelho, e o espaço ocupado por uma chave *hash* em laranja.



Fonte: o autor.

Nota: Repare ao centro que duas caixas laranjas se interceptam. Isto significa que existem vértices que podem ser mapeados para as duas caixas. Contudo, devido a divisão espacial (as linhas espessas pretas), é possível separar as chaves para os *buckets* adequados. O algoritmo ainda permite aumentar a precisão da chave hash, o que diminui o tamanho da caixa laranja, ou aumentar a divisão espacial através do algoritmo da Octree (BRUNET; NAVAZO, 1990).

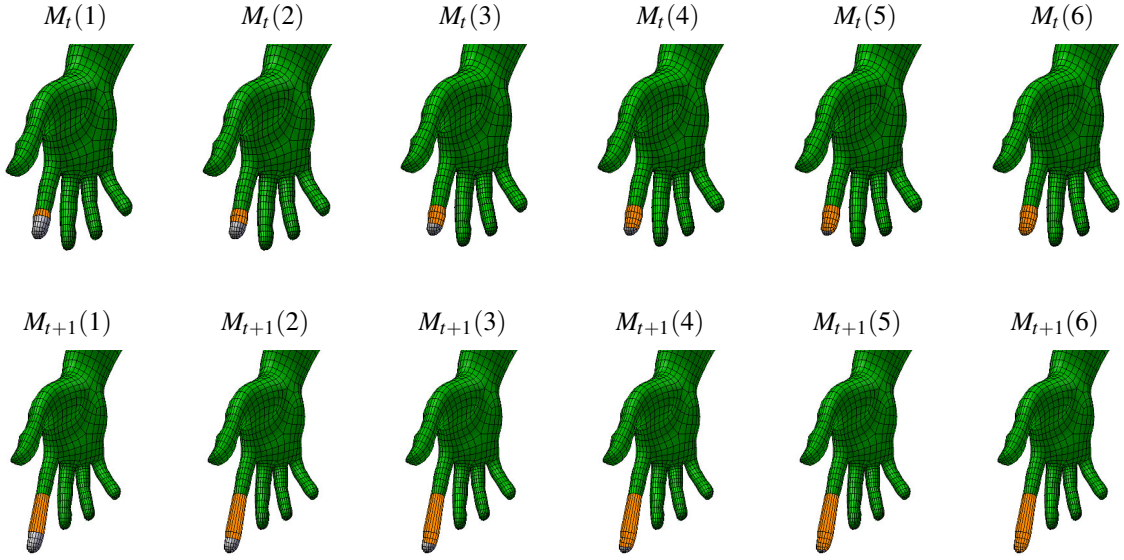
3.4.2 Vértices que se alteram, e não mudam de quantidade

Neste caso, a quantidade de vértices não mudou entre malhas de versões vizinhas, contudo alguns de seus vértices tiveram suas posições alteradas possivelmente deformando outros elementos da malha. Para efetuar o mapeamento, realiza-se um algoritmo que foi nomeado de propagação topológica. Esse algoritmo depende da existência de vértices entre versões diferentes que não tenham se alterado, pois eles são a base para a propagação. Isto implica que existem vértices entre versões diferentes que possuem a mesma chave *hash* (vértices mapeados).

No Algoritmo 2, todo vértice da malha M_{t+1} que pertence ao *Changeset* é inserido no conjunto Q (linha 1), em que t representa a versão da malha. Busca-se, então, por todos os vizinhos de cada vértice v em Q que já tenham sido mapeados (linha 3; v é um vértice da fronteira), e mapeia-se tal vértice v à malha M_t pelo seu vizinho ou vizinhos mapeados (linha 4; 1 vizinho para faces quadrangulares e 2 vizinhos para faces triangulares), i.e., o vizinho ou vizinhos mapeados em M_t possuem um único vizinho u ainda sem correspondência em M_{t+1}, deste modo cria-se uma ligação entre u e v. Essa ligação é um tipo de inter-referência (entre malhas), não autorreferência (mesma malha) que é explicado na próxima seção.

No caso em que a face á qual o vértice v pertence é um quadrilátero, existe apenas um vizinho associado para ser mapeado, contudo, no caso em que a face é um triângulo, existem dois vizinhos associados para o mapeamento. A ligação entre vértices não associados é repetida sucessivamente até que todos os vértices tenham sido mapeados e Q encontre-se vazio (linha 5). A Figura 3.4 exibe visualmente o processo de mapeamento de uma malha M_t, uma mão aberta, e

Figura 3.4 – Em verde pode-se observar vértices já mapeados entre M_t e M_{t+1}. Em cinza, os vértices ainda não mapeados entre as malhas. Em seguida a propagação topológica acontece gradualmente em 6 passos neste caso, os vértices adicionados ao mapeamento são exibidos em laranja.



Fonte: o autor.

sua versão modificada, M_{t+1}, uma mão com um dedo longo.

Como os vértices são mapeados corretamente e em tempo polinomial, assim também o são arestas e faces através dos vértices. Dois vértices associados corretamente entre M_t e M_{t+1} determinam uma aresta, assim como um conjunto de arestas mapeadas corretamente determina uma face. Esta associação só é possível devido a não mudança da quantidade de vértices na malha (o caso de deleção de uma face ou aresta está explicado na próxima seção como caso trivial).

Algoritmo 2: Mapeia os vértices de um *Changeset* entre versões próximas.

```
1 Q ←vértices(Changeset(M_{t+1}));
2 enquanto v ∈ Q faça
3     se ∃ (vizinhos(v)) ∈ M_t então
4         mapeia(v, M_t);
5         Q ← Q − {v};
6     fim
7 fim
```

3.4.3 Vértices que se alteram, e mudam de quantidade

Neste caso, a quantidade de vértices muda entre malhas de versões vizinhas, e alguns de seus vértices tiveram suas posições alteradas possivelmente deformando outros elementos da malha. Para efetuar o mapeamento, ainda é feito a propagação topológica. Contudo, desta vez, os elementos pertencentes a uma malha M_{t+1} podem não encontrar um elemento correspondente a si em M_t, isto é, arestas podem ser mapeadas tanto para arestas quanto para vértices, e faces podem ser mapeadas para vértices, arestas ou faces. Esses casos determinam situações chamadas de autorreferência, em que elementos de uma malha apontam para seus próprios elementos com a exigência de que tais elementos já tenham sido mapeados.

Este novo tipo de ligação é esperado, já que o número de elementos mudou de uma versão a outra por uma exclusão ou inserção de elementos. O objetivo desta referência é mapear os parâmetros das funções *API*s que são removidas da subárvore gerada final quando os parâmetros associados não correspondem ao esperado (i.e., uma função *API* tenta atuar sobre um elemento da malha que não existe mais).

Nas subseções seguintes, lista-se algumas das operações mais comuns efetuadas em um modelador com o resultado do mapeamento para demonstrar este efeito: Deleção (Subseção 3.4.3.1), Inserção (Subseção 3.4.3.2), Extrusão (Subseção 3.4.3.3), Corte de Laço (Subseção 3.4.3.4), Subdivisão (Subseção 3.4.3.5), Bevel (Subseção 3.4.3.6), Duplicação (Subseção 3.4.3.7), e Criação de Buracos (Subseção 3.4.3.8). A lista completa de comandos abordada pelo algoritmo encontra-se no Anexo B.

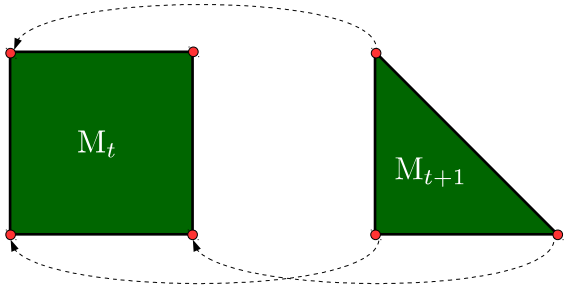
3.4.3.1 Operação de Deleção

Nesta operação, um ou mais vértices são excluídos da malha. Esta é uma das operações de associação mais triviais. Os vértices de M_{t+1} preservam todas as suas correspondências com os em M_t, e, como o Conjunto de Mudanças (*Changeset*) de M_{t+1} é vazio, não é preciso continuar a mapear o que já não existe mais. Neste caso, um ou mais elementos permanecem sem associação (Figura 3.5).

3.4.3.2 Operação de Inserção

Nesta operação, um ou mais vértices são inseridos na malha. Os vértices de M_{t+1} preservam todas as suas corespondências com os em M_t, contudo não há vizinhos destes últimos

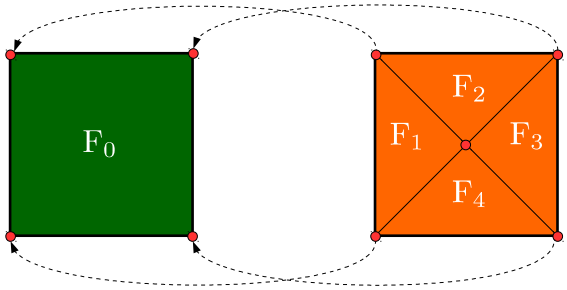
Figura 3.5 – Uma operação de deleção mapeada pelo algoritmo.



Fonte: o autor.

Nota: O vértice excluído não gera um Conjunto de Mudanças na nova malha. Em verde, os elementos que foram mapeados de uma malha M_t para sua versão seguinte M_{t+1}; note que o mapeamento de faces permanece, pois todos os vértices da nova face estão inclusos na face anterior. As setas demonstram o mapeamento efetuado pelo algoritmo.

Figura 3.6 – Uma operação de deleção mapeada pelo algoritmo.



Fonte: o autor.

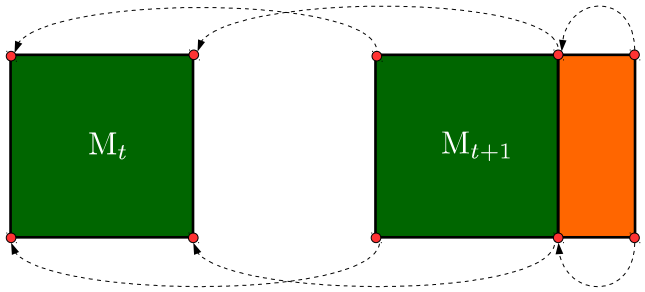
Nota: O vértice excluído não gera um Conjunto de Mudanças na nova malha. Em verde, os elementos que foram mapeados de uma malha M_t para sua versão seguinte M_{t+1}; note que o mapeamento de faces permanece, pois todos os vértices da nova face estão inclusos na face anterior. As setas demonstram o mapeamento efetuado pelo algoritmo.

sem associação já que o Conjunto de Mudanças (*Changeset*) é vazio para M_t. Neste caso, os vértices novos são associados com seus próprios vizinhos em M_{t+1} que já foram mapeados (autorreferência). Na Figura 3.6, observa-se a adição de um único vértice central, e qualquer que seja o mapeamento efetuado, duas faces são mapeadas para F_0 e as outras duas para arestas.

3.4.3.3 Operação de Extrusão

Nesta operação, uma aresta ou face gera um novo polígono ou poliedro respectivamente. Os vértices novos gerados em M_{t+1} possuem vizinhos com correspondência em M_t, contudo não há vizinhos destes últimos sem associação já que o Conjunto de Mudanças (*Changeset*) é vazio. Neste caso, os vértices novos são associados com seus próprios vizinhos em M_{t+1} que já foram mapeados por autorreferência, Figura 3.7.

Figura 3.7 – Uma operação de extrusão mapeada pelo algoritmo.



Fonte: o autor.

Nota: Em verde, os elementos que foram mapeados de uma malha M_t para sua versão seguinte M_{t+1}. Em laranja o Conjunto de Mudanças resultante de uma operação de extrusão em M_t. As setas demonstram o mapeamento efetuado pelo algoritmo.

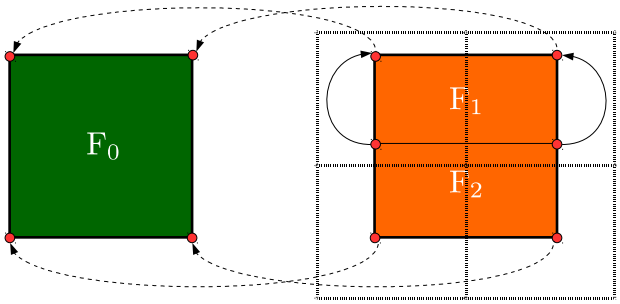
*3.4.3.4 Operação de Corte de Laço (*loopcut*)*

Um laço de arestas é uma sequência de arestas conexas cujo vértice inicial da primeira aresta é igual ao vértice final da última aresta na sequência. Nessa operação, uma ou mais faces que contêm um laço de arestas é cortado gerando-se pelo menos duas novas faces, i.e., são inseridas novas arestas que dividem o laço produzindo novas faces. Novamente, os vértices novos gerados em M_{t+1} possuem vizinhos com correspondência em M_t, contudo não há vizinhos destes últimos sem associação já que o Conjunto de Mudanças (*Changeset*) é vazio. Neste caso, os vértices novos são associados com seus próprios vizinhos em M_{t+1} que já foram mapeados usando como referência também a divisão espacial hierárquica da região, Figura 3.8. É suficiente para realizar uma associação de faces que todos os seus vértices estejam mapeados conforme pode-se observar na Figura 3.9, mesmo com a divisão espacial um vértice permanece sem associação. Tomando a associação da figura, formam-se duas faces triangulares. Contudo, ambas as faces possuem todos os vértices mapeados, e, portanto, F_1 e F_2 apontam para F_0.

3.4.3.5 Operação de Subdivisão

Nesta operação uma ou mais faces são subdivididas pelo ponto médio para aumentar o grau de suavização ou detalhe de uma malha. O ponto médio de cada aresta é unido ao ponto médio da face segundo o algoritmo de Catmull-Clark (CATMULL; CLARK, 1978). Quando este algoritmo atua em parte da malha, é possível efetuar o mapeamento dos novos vértices para seus vizinhos já mapeados, uma vez que não possuem correspondência em M_t de maneira similar ao corte de laço. De fato, o processo pode ser visto como dois ou mais cortes de laço quando os vértices não se movem como pode ser visto na Figura 3.10.

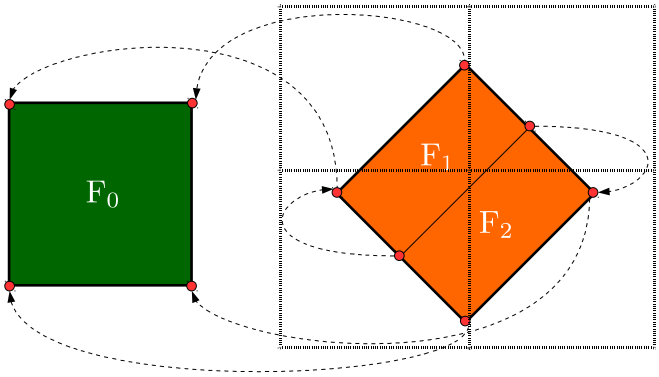
Figura 3.8 – Uma operação de corte de laço mapeado pelo algoritmo.



Fonte: o autor.

Nota: Em verde, os elementos que foram mapeados de uma malha para sua versão seguinte. Em laranja o Conjunto de Mudanças resultante de uma operação de corte de laço em M_t. As setas demonstram o mapeamento efetuado pelo algoritmo. Apenas F_2 é mapeada para F_0, F_1 é mapeada para uma aresta devido a associação dos vértices.

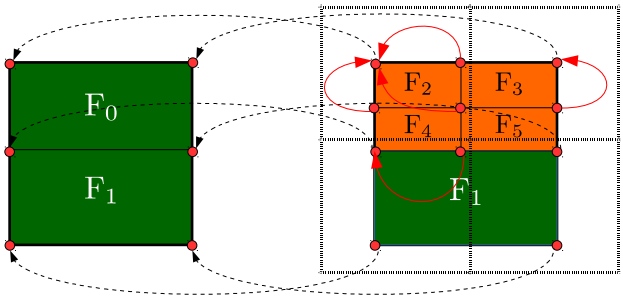
Figura 3.9 – Uma operação de corte de laço mapeado pelo algoritmo.



Fonte: o autor.

Nota: Em verde, os elementos que foram mapeados de uma malha para sua versão seguinte. Em laranja o Conjunto de Mudanças resultante de uma operação de corte de laço em M_t. As setas demonstram o mapeamento efetuado pelo algoritmo. Ambas as faces F_1 e F_2 são mapeadas para F_0, já que todos os vértices das faces pertencem a F_0 devido a associação dos vértices.

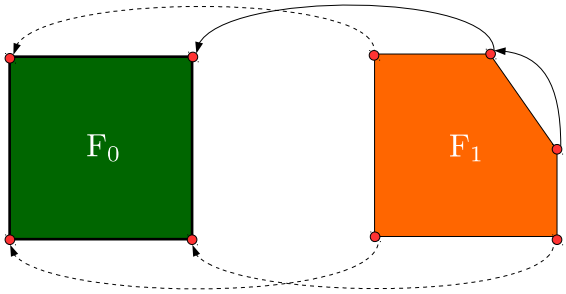
Figura 3.10 – Uma operação de subdivisão mapeada pelo algoritmo.



Fonte: o autor.

Nota: Em verde, os elementos que foram mapeados de uma malha para sua versão seguinte. Em laranja o Conjunto de Mudanças resultante de uma operação de subdivisão em M_t. As setas em vermelho demonstram um possível mapeamento efetuado pelo algoritmo segundo a subdivisão espacial hierárquica. F_5 é mapeada para F_0, F_4 e F_3 são mapeados para arestas, e F_2 é mapeada para um vértice.

Figura 3.11 – Uma operação de bevel mapeada pela algoritmo.



Fonte: o autor.

Nota: Em verde, os elementos que foram mapeados de uma malha para sua versão seguinte. Em laranja o Conjunto de Mudanças resultante de uma operação de bevel em M_t, trocando um vértice por uma aresta. Um dos vértices adicionados é mapeado para o vértice original em M_t, enquanto o outro sofre autorreferência. Isto significa que F_1 ainda é mapeada para F_0 (mesmo sendo um polígono de mais lados), e a aresta nova é mapeada para um vértice.

3.4.3.6 Operação de Bevel

Na operação de Bevel, um elemento é substituído por um elemento de outro tipo de maior grau. Por exemplo, uma aresta é substituída por uma face (conjunto de arestas), ou um vértice por uma aresta (dois vértices). Neste caso, existem em M_t elementos com vizinhos ainda não mapeados. O algoritmo utiliza-se da chave hash primeiro para encontrar estes vizinhos em M_t sem associação (inter-referência), e em seguida, os elementos restantes aplicam autorreferência como demonstrado na Figura 3.11.

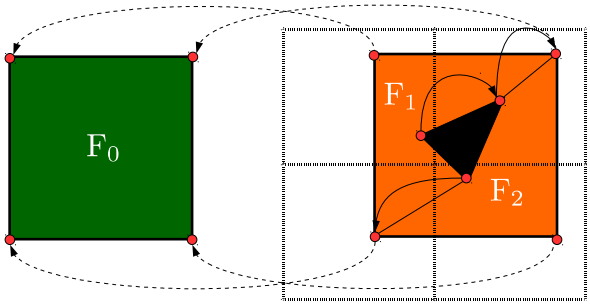
3.4.3.7 Operação de Duplicação

Há várias operações de modelagem que duplicam elementos, e da mesma forma que no caso da subdivisão desde que a malha inteira não seja duplicada, e exista uma conexão entre o elemento duplicado e a malha, é possível efetuar o mapeamento. O processo neste caso funciona como uma extrusão, e a autorreferência é usada de forma incremental por propagação topológica. Caso não exista conexão da malha com os novos elementos, eles formam uma nova malha para todos os efeitos do algoritmo. Isto significa que cada malha possui um conjunto próprio de referências (uma subárvore com origem nesta operação global de duplicação).

3.4.3.8 Operação de Criação de Buracos

Existem várias operações que criam buracos em uma malha. Um buraco é a ausência de um elemento da malha, normalmente uma face, e, portanto, possui ligações com outros

Figura 3.12 – Uma operação de criação de buracos mapeada pela algoritmo.

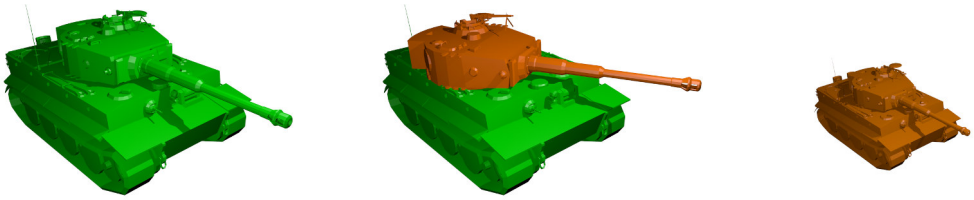


Fonte: o autor.

Nota: Em verde, os elementos que foram mapeados de uma malha para sua versão seguinte. Em laranja o Conjunto de Mudanças resultante de uma operação que criou o buraco em M_t, o pedaço de uma face foi removido, obrigando a inserção de vários elementos para constituir a fronteira do buraco. Os vértices adicionados são mapeados por autorreferência para os vizinhos mais próximos graças a subdivisão espacial. Note que em qualquer caso similar ao da figura, as faces F_2 e F_1 são mapeadas para F_0, pois os vértices só podem ser mapeados para vizinhos espaciais e que possuem uma aresta com ligação topológica.

Figura 3.13 – Em (a), é possível observar a versão t de uma malha de um tanque, em (b), uma operação local foi aplicada à malha M_t, e, em (c), uma operação global foi aplicada a malha M_t. A cor laranja é usada para destacar *Changeset* das malhas possíveis M_{t+1}.

(a) M_t (b) M_{t+1} (c) M_{t+1}



Fonte: o autor.

elementos da malha. O buraco ao mesmo tempo que remove elementos, também pode inserir novos que constituem a fronteira do buraco, o contorno da região removida. A partir deste contorno pode-se encontrar os elementos ainda conectados a malha M_t e efetuar o autorreferência como é possível observar na Figura 3.12.

3.5 Classificação das operações

Uma vez que o mapeamento dos vértices tiver sido concluído, cada operação em um nó da árvore de versões é classificada de acordo com o seu *Changeset* (denotado como região $R_U(OP_t)$), i.e., a diferença entre duas malhas. As operações podem ser classificadas como *globais* ou *locais* (Figura 3.13). Se o Conjunto de Mudanças, *Changeset*, possui a malha M_t ou M_{t+1}, em que t indica sua versão, a operação recebe o rótulo de global, e o rótulo de local quando não possui nenhuma das duas malhas.

O *Changeset* é computado deterministicamente pelo método apresentado e isto garante que não existem operações inválidas na subárvore especializada (falsos positivos), i.e., atuam em regiões diferentes da seleção inicial. Note que para esta computação não é necessário conhecer o mapeamento dos vértices no *Changeset*, apenas o mapeamento dos vértices em comum à ambas as malhas (fora do Conjunto de Mudanças).

Para identificar um *Changeset*, como já foi dito, o algoritmo usa *geometric hashing* (WOLFSON; RIGOUTSOS, 1997) com uma precisão de 4 casas decimais para gerar suas chaves *hash*. Ao atravessar um nó qualquer t sem perda de generalidade, duas tabelas *hash* de correspondência são preenchidas: uma para faces e a outra para vértices. No próximo nó $t+1$, os elementos com chaves conflitantes, que caem no mesmo *bucket*, indicam que vértices e faces devem ser comparados para verificar semelhanças, enquanto duas novas tabelas são geradas.

Os vértices semelhantes possuem necessariamente chave *hash* igual. Os vértices que não possuem índice *hash* na tabela, ou que, após a comparação dos elementos contidos nos respectivos *buckets*, não encontram elementos semelhantes fazem parte obrigatoriamente do *Changeset*. Este é, então, armazenado por meio de um código de cores (conforme paleta da Figura 4.10) em memória para acesso futuro.

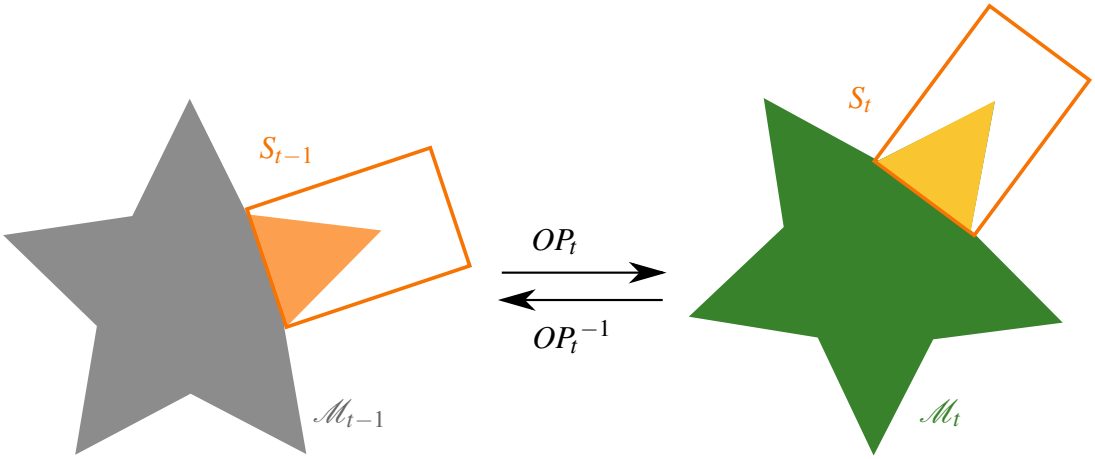
Esse processo de classificação também pode ser feito através da seleção existente no interior de um nó, os seus parâmetros (que não é a seleção de entrada ou propagada). Observe, contudo, que o *Changeset* é diferente da região selecionada por uma operação em um nó e que o algoritmo proposto usa *Changesets*, logo, os resultados obtidos pelo algoritmo podem ser diferentes (caso a entrada seja alterada). Por exemplo, o movimento de uma face pode afetar todas as faces vizinhas (para efetuar este movimento houve primeiro a seleção da face, armazenada como um parâmetro dentro do nó da árvore de versões), e, o algoritmo proposto nesta tese considera todas as faces alteradas, e não apenas a face que foi selecionada, que corresponde a seleção armazenada no nó.

3.6 Propagação da seleção

Após a classificação das operações, realiza-se a fase de propagação da seleção. Propagar a seleção significa copiar a seleção de uma versão para todas as outras realizando modificações ou não no seu posicionamento e formato.

A seleção determina uma região de interesse que o usuário deseja que seja perseguida (*tracking*) para gerar um tutorial ou desfazer uma operação, e apresenta-se como uma malha

Figura 3.14 – Evolução de uma seleção em uma única operação na malha (Seção 3.6).



Fonte: o autor.

Nota: Assumindo que $t-1$ é o pai do nó t sem perda de generalidade, sempre que a operação $OP_t : M_{t-1} \rightarrow M_t$ é global e modifica a geometria, a seleção correspondente S_{t-1} é transformada em S_t pela mesma operação. Quando a propagação ocorre no sentido inverso da geração, a operação OP_t^{-1} é usada para transformar S_t em S_{t-1}. A paleta de cores é a mesma que a Figura 4.10 usa.

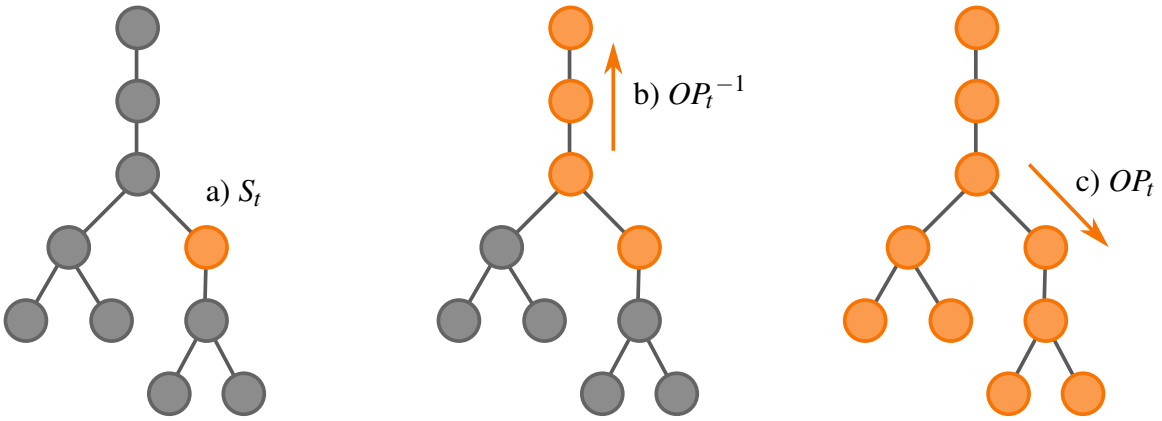
simples dotada de seis faces quadrangulares, doze arestas e oito vértices (Figura 4.3). Ela é feita pelo usuário através de uma *Bounding Box* (CHANG *et al.*, 2011) aplicada sobre um subconjunto da versão de um modelo após a classificação das operações (Seção 3.5).

A seleção é afetada por operações globais que alteram a geometria da malha – translação, rotação, cisalhamento, escala, etc. (transformações rígidas) – e isto significa que elas alteram a posição ou o formato da seleção (Figura 3.14). Isto permite que o posicionamento espacial da seleção busque (*tracking*) com sucesso a região desejada pelo usuário, e é feito de acordo com o histórico de operações armazenado na ACV.

Deste modo, a seleção é propagada uma vez em direção da raiz, e, em seguida, na direção das folhas, transmitindo a seleção inicial para todos os nós da árvore de versões (Figura 3.15) por busca em profundidade sem revisitar nós. Note que ao propagar a seleção na direção da raiz é necessário calcular a matriz inversa das operações que modificam a seleção. A matriz identidade é usada para operações que não possuem uma matriz de transformação.

As operações locais não alteram a seleção, e isto pode implicar na perda da sua posição perseguida. Por exemplo, se a seleção é feita sobre um membro de um modelo que foi rotacionado isoladamente, a seleção propagada pode permanecer ocupando um espaço vazio. Para solucionar este problema, é o suficiente que o usuário faça uma seleção grande o suficiente ou que aplique o processo duas vezes, aumentando a região de seleção na segunda vez.

Figura 3.15 – A propagação de uma seleção determinada previamente (Seção 3.6): nós visitados estão em laranja e nós não-visitados em cinza. O passo de entrada é uma seleção determinada no nó t (a). A partir deste ponto, a seleção espacial é propagada na direção da raiz aplicando o inverso das operações globais quando necessário (b) até alcançar a raiz. Em seguida, a seleção é propagada em direção de todas as folhas usando, sempre que necessário, as mesmas operações geométricas aplicadas a malha (c).



Fonte: o autor.

3.7 Testes de interseção

Durante a propagação da seleção alguns testes de interseção são usados. Os testes de interseção identificam quando uma seleção possui alguma região em comum com um Conjunto de Mudanças (*Changeset*). Não é necessário saber que elementos estão no conjunto de interseção. Quando este conjunto não é vazio significa que a operação efetuada no nó gerador do Conjunto de Mudanças faz parte da subárvore que é extraída.

Efetuam-se três tipos de testes para identificar se existe um conjunto de interseção não vazio (ERICSON, 2004):

a) Vértice dentro de Volume;

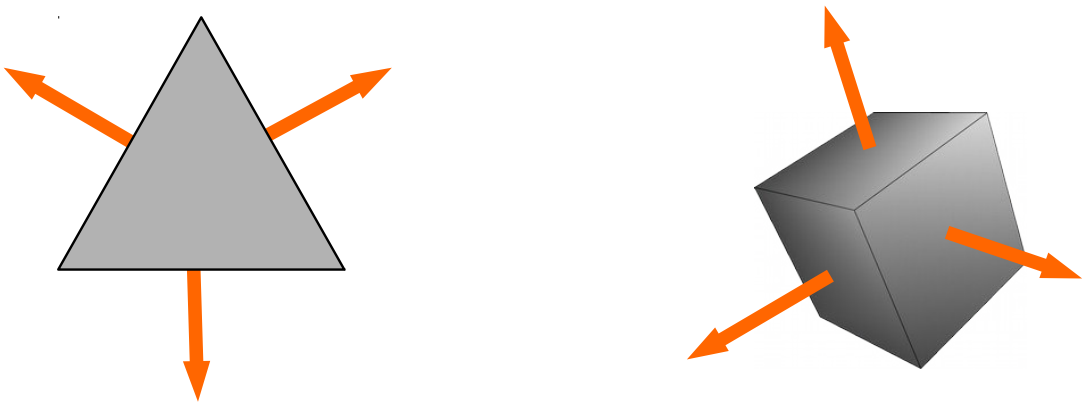
b) Aresta intercepta Face; e

c) Face intercepta Face.

Esses testes são detalhados nas subseções a seguir: Vértice dentro de Volume (Subseção 3.7.1), Aresta intercepta Face (Subseção 3.7.2), e Face intercepta Face (Subseção 3.7.3).

Figura 3.16 – Em (a), é possível observar uma face triangular em cinza com sua normais em laranja apontando para fora da face. As normais, no caso 2D, são perpendiculares as arestas do polígono que forma a face. Já em (b), um hexaedro é exibido em cinza, com as normais em laranja também apontando para fora de seu interior. As normais, no caso 3D, são ortogonais aos polígonos que formam as faces.

(a) (b)



Fonte: o autor.

3.7.1 Vértice dentro do Volume da Caixa Envoltória

Para calcular se um vértice v está dentro do volume da caixa envoltória, projeta-se o vetor que une este vértice v aos vértices de cada face da caixa nos vetores normais de cada plano da face, e obtém-se o módulo dessa projeção sinalizado. Assumindo sem perda de generalidade que as normais dos planos apontam para fora da caixa envoltória (Figura 3.16), se o sinal da projeção para os seis planos é não-positivo, o vértice encontra-se dentro da Caixa Envoltória.

Para cada normal $\vec{n}$ unitária de um plano da caixa, existe um vértice u deste plano tal que $\vec{w} = v - u$. Efetua-se a projeção de $\vec{w}$ em $\vec{n}$:

$$proj(\vec{w})_{\vec{n}} = \vec{w}.\vec{n}. \tag{3.1}$$

O algortimo 3 demonstra a implementação do teste. Na linha 2, cria-se o vetor que une um vértice u_f da face f ao vértice v. E se, em algum momento, o valor da projeção de $\vec{w_f}$ na normal da face for positivo (Equação 3.1, linha 3), o vértice está fora da caixa.

3.7.2 Aresta intercepta Face da Caixa Envoltória

Para calcular se uma aresta da malha M_t qualquer se intercepta com uma face da caixa, é preciso encontrar o ponto de interseção p da reta de mesma direção que a aresta e um plano que constitui a Caixa Envoltória, tal que p pertença a aresta, e que p esteja dentro do

Algoritmo 3: Verifica se um vértice está na caixa envoltória.

```
1 para cada face f da caixa faça
2 |   w_f ← v − u_f;
3 |   se proj(w_f)_{n_f} > 0.0f então
4 |   |   retorna fora_caixa;
5 |   fim
6 fim
7 retorna dentro_caixa;
```

quadrângulo que determina a face (o mesmo teste de Vértice dentro de Volume).

Para realizar o teste, é usada a equação paramétrica da reta (Equação 3.2), tal que o coeficiente $\delta \in [0,1],\ \delta \in \mathbb{R}$, o é o vértice de origem e d é o vertice de destino de um vetor formado pela aresta com uma orientação qualquer, e p é o ponto de interseção da reta na face:

$$p = o + \delta(d - o). \tag{3.2}$$

A equação usada para os seis planos que formam as faces da Caixa Envoltória é a Equação 3.3 , em que $\vec{n}$ é a normal do plano da face, e u é um vértice pertence a face, e p é o ponto de interseção da reta na face:

$$\vec{n}.(p - u) = 0. \tag{3.3}$$

Substituindo a Equação 3.2 na Equação 3.3 obtém-se a Equação 3.4, usada para obter o valor do coeficiente δ. Com este valor, é possível aplicá-lo na Equação 3.2, e encontrar o ponto de interseção. Uma vez que obtém-se este ponto pode-se aplicar o mesmo Algoritmo 3 para identificar se ele pertence a face da caixa:

$$\begin{aligned} \vec{n}.(o + \delta.(d - o) - u) &= 0 \\ \vec{n}.o + \delta.\vec{n}.(d - o) - \vec{n}.u &= 0 \\ \delta.\vec{n}.(d - o) &= \vec{n}.u - \vec{n}.o \\ \delta &= \frac{\vec{n}.(u - o)}{\vec{n}.(d - o)}. \end{aligned} \tag{3.4}$$

O Algoritmo 4 demonstra o processo. Na linha 2, exibe-se o cálculo de c conforme a Equação 3.4, e se este estiver no intervalo $[0,1]$, encontra-se o ponto de interseção na linha 4. Da linha 5 em diante, é o mesmo processo efetuado pelo Algoritmo 3, porém adaptado para área ao invés de volume.

Algoritmo 4: Verifica se uma aresta intercepta com uma caixa envoltória.

```
1  para cada face f da caixa faça
2      δ ← n⃗.(u − o)/n⃗.(d − o);
3      se (δ ⩾ 0) ∧ (δ ⩽ 1) então
4          p ← o + δ(d − o);
5          para cada aresta a da caixa faça
6              w⃗_a ← p − u_a;
7              n⃗_a ←⊥ a⃗;
8              se proj(w⃗_a)_{n⃗_a} > 0.0f então
9                  retorna intercepta_caixa;
10             fim
11         fim
12     fim
13 fim
14 retorna nao_intercepta_caixa;
```

3.7.3 Face da Malha intercepta Face da Caixa Envoltória

Para calcular se uma face da malha M_t se intercepta com uma face da caixa envoltória, testa-se se alguma aresta da face se intercepta com a Caixa Envoltória, e se alguma aresta da Caixa Envoltória se intercepta com a face.

O algoritmo neste caso repete o Algoritmo 4 para cada aresta de uma face malha M_t e em seguida repete-o para todas as arestas da Caixa Envoltória contra a face da malha M_t, em que t é a versão da malha. Para não tornar tais testes computacionalmente caros, usa-se a divisão hierárquica do espaço para limitar os testes para dentro da região espacial que a Caixa Envoltória ocupa.

3.8 Geração da subárvore

A subárvore especializada é gerada após a Propagação da seleção (Seção 3.6) por toda a árvore, enquanto é testado que vértices ou faces do *Changeset* estão dentro ou fora da Caixa Envoltória correspondente para cada nó; e, se necessário, é testado também a intersecção entre as faces (denotada como região $R_{S_t}(OP_t)$). Os cálculos de intersecção são feitos de acordo com Ericson (2004) e foram explicados nas seções anteriores.

Um nó é inserido na subárvore especializada se produz um conjunto de interseção não-vazio entre o *Changeset* e o volume definido pela seleção correspondente (a Caixa Envoltória). Uma vez que toda a árvore é percorrida, a subárvore é obtida, e é possível exibir a versão original da árvore principal e sua subárvore especializada (Figura 4.4).

A subárvore especializada contém para uma mesma região selecionada todas as maneiras possíveis de construir tal região com todas as versões possíveis. É através delas que é possível extrair tutoriais e realizar o Desfazer Localizado. A subárvore é a coleção dos resultados de operações que afetam a região de interesse de um usuário.

3.9 Desfazer Localizado

O processo de desfazer operações baseadas em regiões determinadas sobre uma malha a partir de uma árvore de versões traduz-se em remover nós locais obtidos a partir da subárvore gerada pelo algoritmo exposto. Uma vez que os nós tenham sido removidos de forma automática ou por um usuário, a árvore de versões executa os comandos dentro do modelador original que formou a malha pela Sequência de Construção em que o usuário se encontra atualmente de forma a gerar uma nova versão da malha (nesta tese, os comandos são executados através da *Maya Embedded Language*/linguagem intríseca do Maya (*MEL*), Maya Embedded Language; AUTODESK,2016b).

Qualquer operação de Desfazer Localizado na subárvore torna o nó transparente na árvore principal: o pai do nó desfeito se torna o pai dos seus filhos desta forma. Se necessário, interação com a subárvore gera um aviso de possível perda de dados, caso exista tentativa de remoção de nós classificados como globais (somente nós locais podem ser removidos). Nós globais são nós de origem para mapeamentos de vértices e, portanto, para atualização de parâmetros, sua remoção pode ocasionar efeitos imprevisíveis.

Uma vez que uma ou mais operações são desfeitas, todas as Sequência de Construção que possuam nós que foram removidos são executadas para gerar as novas versões (além da Sequência de Construção que foi usada pelo usuário). É desnecessário reaplicar todas as operações da árvore obrigatoriamente, já que nem todas as versões podem sofrer modificações.

O algoritmo permite a remoção de operações em qualquer ordem; ao desfazer uma operação de uma árvore de versões pode ser necessário remapear vértices ou faces que não existem mais antes de reaplicar uma operação do histórico dependente de tais parâmetros. Para realizar isto, usa-se dos mapeamentos explicados na Seção 3.4 para identificar os elementos, e a propriedade de transição de referências para transmitir as referências existentes entre todas as operações locais ou entre as operações globais e locais. Por exemplo, se os vértices da malha M_t são mapeados para M_{t+1}, e se os vértices da malha M_{t+1} são mapeados para M_{t+2}, então, no caso da remoção do nó que gera M_{t+1}, é possível mapear os vértices de M_t para M_{t+2}.

Também é possível que seja necessário remover operações que são dependentes da operação desfeita; para isso todos os nós posteriores a uma operação desfeita são removidos da subárvore especializada. Por exemplo, suponha que o usuário original tenha pintado a camisa de um modelo 3D com uma cor qualquer, se as operações que criam a camisa são removidas, não é mais possível atribuir uma cor a um elemento que não existe, e portanto tal operação também necessita ser removida.

3.10 Algoritmo

Esta seção apresenta a versão formal do algoritmo. Os algoritmos para extrair uma subárvore (Algoritmo 5) e para efetuar o Desfazer Localizado (Algoritmo 6) são descritos aqui.

Dado um conjunto universo $U = \mathbb{R}^3$, assuma que uma única seleção espacial determinada S_t exista tal que $\forall_{i \leq t} : U \supset S_i,\ M_i(F_i, V_i)$, em que $t > 0$; $t, i \in \mathbb{N}$ representam as versões das malhas; S_i é inicialmente uma Caixa Envoltória alinhada com os eixos canônicos do espaço $\mathbb{R}^3$; M_i é uma malha com um conjunto de faces F_i assim como um conjunto de vértices V_i; e $M_0 = \emptyset$.

Em seguida, assumindo que $i-1$ é o pai do nó i sem perda de generalidade, e $y \in \{F, V\}$, define-se: a árvore de versões B como a união de todas as operações $B = \bigcup_{i=0}^{t} OP_i$; uma operação $OP_i : M_{i-1} \rightarrow M_i$ como

$$OP_i = \{y \in \mathscr{P}(\Lambda) \mid \Lambda = \{\forall t : \bigcup \{D(y_i), I(y_i), T(A_i, y_{i-1})\}\}\}, \tag{3.5}$$

em que $\mathscr{P}(\Lambda)$ é o conjunto das partes de Λ, $D : M_{i+1} \leftarrow M_i - \{y\}$ e $I : M_{i+1} \leftarrow M_i \cup \{y\}$ são respectivamente as operações de deleção e a inserção de elementos; A_i é uma matriz, $T : A_i y_i = y_{i+1}$ é a transformação $T : \mathbb{R}^3 \rightarrow \mathbb{R}^3$; e a região afetada pela seleção espacial como

$$R_{S_i}(OP_i) = \bigcup_{t=i}^{i+1} \{y_i \in M_i \mid (y_i \neq y_{i+1}) \wedge ((F_i \cap S_i) \vee (V_i \in S_i))\}. \tag{3.6}$$

Todas as operações dentro de uma árvore de versões B são analisadas de acordo com o seu *Changeset* correspondente, a região do Conjunto Universo R_U (Equação 3.6), nas linhas 2 à 9. Quando as operações afetam toda malha de uma determinada versão M_i – transformações de corpo rígido, escala, cisalhamento, etc. – elas são classificadas como globais; caso contrário elas são classificadas como locais. O método proposto propaga a seleção de entrada (linha 10) para os nós anteriores em uma Sequência de Construção (linhas 13 à 22); ele retrocede na árvore de versões até a raiz da ACV, e aplica a matriz inversa $OP_{nó}^{-1}$ em $S_{nóAnterior}$ se o nó corrente $S_{nó}$ for um nó global (A_i é igual a identidade por *default*). Finalmente, nas linhas 24 à 38, o método

termina de criar a seleção espacial para nós que não foram visitados (ele avança na árvore de versões), determina que regiões se interceptam com a seleção, a região da seleção $R_{S_{nó}}$, na linha 34 (Equação 3.6), e produz o conjunto B_{sub}.

Algoritmo 5: Calcula a subárvore $B_{sub} \subset B$.

```
1  início
2      para cada nó de B faça
3          se R_U(OP_nó) ⊃ M_nó então
4              tipo(OP_nó) ← global;
5          fim
6          senão
7              tipo(OP_nó) ← local;
8          fim
9      fim
10     S_i ← BB de uma determinada seleção sobre uma versão (nó i);
11     nó ← i;
12     nóAnterior ← pai(nó);
13     enquanto nó ≠ raiz faça
14         se tipo(OP_nó) = global então
15             S_nóAnterior ← OP_nó^-1(S_nó);
16         fim
17         senão
18             S_nóAnterior ← S_nó;
19         fim
20         nó ← nóAnterior;
21         nóAnterior ← pai(nó);
22     fim
23     nó ← raiz;
24     enquanto nó ≠ i faça
25         se S_nó = ∅ então
26             nóAnterior ← pai(nó);
27             se tipo(OP_nó) = global então
28                 S_nó ← OP_nó(S_nóAnterior);
29             fim
30             senão
31                 S_nó ← S_nóAnterior;
32             fim
33         fim
34         se R_S_nó(OP_nó) ≠ ∅ então
35             B_sub ← B_sub ∪ OP_nó;
36         fim
37         nó ← nó + 1;
38     fim
39 fim
```

O Algoritmo 6 demonstra o processo automático para o Desfazer Localizado. Todos os nós locais da subárvore são removidos tornando-os invisíveis para a árvore principal (linhas 2 e 3). Em seguida, as Sequências de Construção são extraídas da subárvore e executadas progressivamente (linhas 6, 7 e 8). O processo de extração é feito através do algoritmo de busca de profundidade (CORMEN *et al.*, 2001).

Algoritmo 6: Remove operações OP_i locais da subárvore B_{sub}.

```
1  para cada nó de B_sub faça
2  |  se tipo(OP_nó) = local então
3  |  |  OP_nó.visivel ← Falso;
4  |  fim
5  fim
6  enquanto (seq ← extraiSequênciaDeConstrução(B)) ≠ ∅ faça
7  |  para OP_nó ∈ seq faça
8  |  |  executaAPI(OP_nó);
9  |  fim
10 fim
```

3.11 Otimização do algoritmo

O algoritmo foi então otimizado da seguinte maneira: cada uma das etapas do algoritmo age em pares de nós ou em único nó de forma independente. O cálculo de Changesets depende da informação contida em pares de nós, enquanto a Classificação das Operações e o Cálculo das Interseções depende da informação contida um único nó. Assim, o algoritmo é dividido no máximo de processadores possível por nós dois a dois, ou um a um, para incrementar a velocidade do algoritmo. A etapa da Propagação da Seleção é feita por um único processador até alcançar a raiz, e da raiz para folhas, embora este processo possa ainda ser otimizado.

Para executar uma Sequência de Construção no modelador original (por exemplo, o Maya) para gerar uma nova versão, todas as operações são efetuadas no mesmo processador. Isto é devido a dependência entre operações de uma sequência. Contudo, Desfazeres Localizados diferentes são executadas em processadores diferentes.

No próximo capítulo (Capítulo 4), os resultados obtidos são apresentados.

4 RESULTADOS

Nesta capítulo, os detalhes de implementação são descritos (Seção 4.1), e os resultados obtidos são apresentados para Tutoriais (Seção 4.2), assim como para o Desfazer Localizado (Seção 4.3). A técnica é analisada por sua eficiência no mapeamento por tabelas *hash* (Seção 4.4), que representa a principal etapa para o cálculo do Conjunto de Mudanças (*Changesets*), e por sua eficiência na paralelização implementada (Seção 4.5), que determina sua velocidade total. Em seguida, o método é comparado com uma das técnicas que representam o estado da arte na área (Seção 4.6). Por fim, as limitações correntes são discutidas (Seção 4.7).

4.1 Detalhes de Implementação

O modelador utilizado para criar os modelos desta tese foi o Maya versão de 2011 (AUTODESK, 2016a), enquanto o Vistrails Provenance Explorer Plugin for Maya versão de 2009 (N.Y.U., 2009) foi responsável por criar uma árvore de controle de versões enquanto o artista desenhava os modelos. A Figura 4.1 mostra a interface deste processo, onde cada operação efetuada no Maya adiciona um novo nó na árvore do Vistrails (N.Y.U., 2005). Entretanto, qualquer outro modelador e/ou árvore com as características descritas no trabalho poderiam ser usados pelo método.

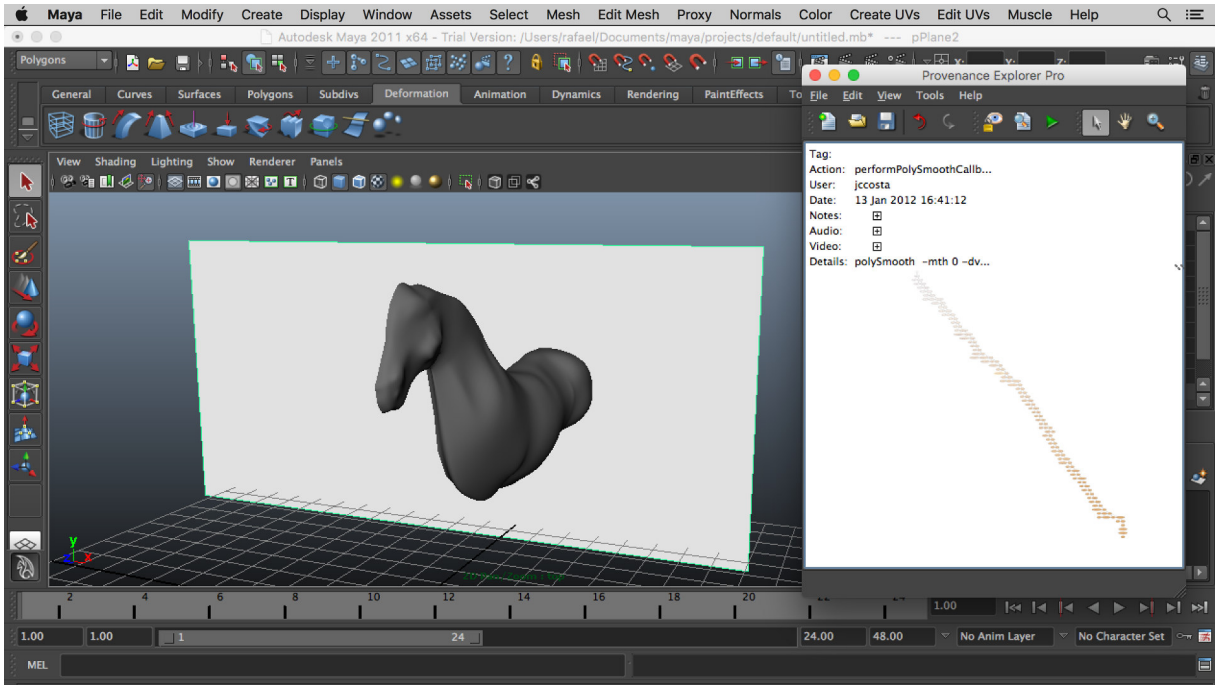
As funções API do Maya são armazenadas em arquivos com extensão VT do Vistrails Provenance Explorer Plugin for Maya, que possuem uma descrição das operações efetuadas na malha em alto nível em um arquivo *eXtensible Markup Language*/linguagem de marcação extendida (*XML*) (chamado de *vistrail*). Abaixo pode ser visto um trecho do arquivo *XML*, *vistrail*, como exemplo para uma operação de escala no nó 1111:

Código-fonte 1 – Operação salva pelo plugin do VT

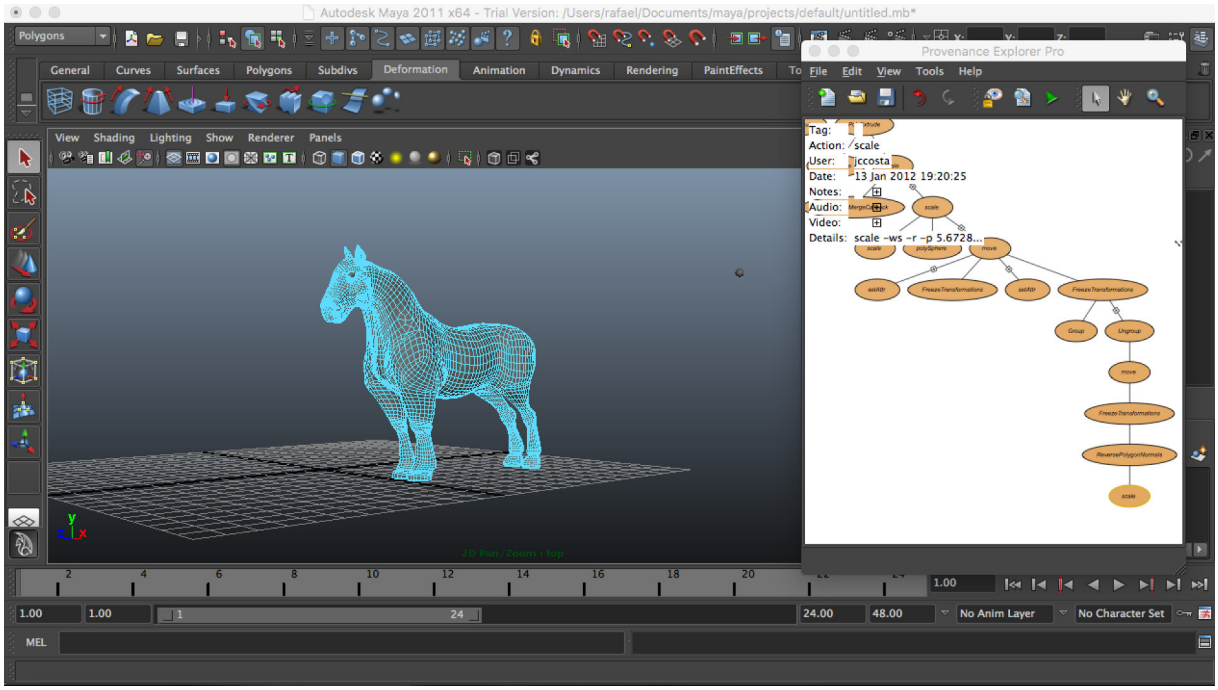
```
1 <action id="1111" prevId="1110" type_op="local">
2 <annotation id="3198" key="__description__" value="scale "/
     >
3 <add id="1110" objectId="1110" what="plugin_data">
4 <plugin_data data="18366043:18428456" id="1110"/>
5 </add>
6 </action>
```

Figura 4.1 – O modelador Maya com o Vistrails Provenance Explorer Plugin.

(a)



(b)



Fonte: o autor.

Nota: Uma janela é exibida dentro do modelador que exibe a árvore de versões de um arquivo de extensão Vistrails (VT) (Vistrails) (N.Y.U., 2005). O plugin possui as funções de um *player* de vídeo parar (*stop*), avançar (*play*), e avanço rápido ($fast - forward$). Na figura (a), toda a árvore de versões com 1196 nós é exibida com um nó selecionado ao seu meio, e, na Figura (b), a versão final da malha do cavalo é exibida com um zoom na folha correspondente.

O arquivo VT também possui as funções API do Maya em um arquivo com os detalhes de baixo nível (elementos da malha, seleções, posicionamento de câmera, etc.), chamado de *data* (seus detalhes estão no Anexo A). A etiqueta *plugin_data* do arquivo *XML* contém os *bytes* que denotam o início e o fim da função API no arquivo *data* separados por dois-pontos (:).

O Vistrails Provenance Explorer tira *snapshots* da malha (arquivos que contém a malha com seus elementos) para acelerar o processo de visualização das malhas, mas, para pode efetuar a técnica apresentada nesta tese, os nós com estes *snapshots* são removidos, e as malhas extraídas executando todas as funções API no modelador, de modo a gerar uma malha para cada operação efetuada (o código para realizar a extração foi inserida dentro do código do plugin). Essas malhas são armazenadas em arquivos OBJ, e, posteriormente, carregadas para a memória *Random Access Memory*/memória de acesso aleatório (*RAM*), onde permanecem até o final da execução do algoritmo.

Os Conjuntos de Mudanças ou *Changesets* são gravados na memória, mas também são armazenados em arquivos de cores (extensão COLOR) que indicam as faces afetadas por cada operação para depois poderem reproduzir as imagens usadas pela tese no Blender (BLENDER, 2016). Cada cor é representada por uma letra do alfabeto de modo que *G*, *R*, *B* e *W* representa respectivamente os quatro estados de um elemento: *Gray* (cinza), elementos da Caixa Envoltória que não fazem parte do Conjunto de Mudanças, *bRown* (marrom), elementos do Conjunto de Mudanças que pertencem a Caixa Envoltória, *Black* (preto), elementos do Conjunto de Mudanças que não pertencem a Caixa Envoltória, e *White* (branco), elementos sem mudanças. Por exemplo, um arquivo de cor com *BBBBBBBBBB*, indica uma malha com 10 faces que pertencem ao Conjunto de Mudanças, mas não à Caixa Envoltória.

O tamanho total das árvores de versão (ACV) considerando o tamanho das malhas para o uso de memória *RAM* é de 1.2 MB para a malha da Espaçonave, 63 MB para a malha do Cavalo, 146 MB para a malha do Bípede[1], 265 MB para a malha do Tubarão[1], 365.6 MB para a malha de Dom Quixote. O tamanho do arquivo VT, com apenas *XML* e dados da função API do Maya armazenados pelo plugin sem os *snapshots* é bem menor[1]: 32 KB para a malha da Espaçonave (92 operações), 1.1 MB para a malha do Cavalo (1196 operações), e 5.4 MB para a malha de Dom Quixote (1648 operações).

O algoritmo cria um arquivo *XML* para a subárvore especializada com a classificação de global ou local para cada nó e as relações de herança – armazenadas em uma tag chamada

[1] modelos obtidos de Dennning (DENNING *et al.*, 2011) não possuem arquivos VT, i.e., as operações do modelador, que pode não ter sido o Maya, não foram salvas.

subtree. Um tutorial é uma Sequência de Construção extraída dessas subárvores especializadas armazenadas no *XML*, como por exemplo, para o nó 184 da subárvore de versão da malha do Cavalo (seleção de todas as pernas):

Código-fonte 2 – Operação capturada pela subárvore

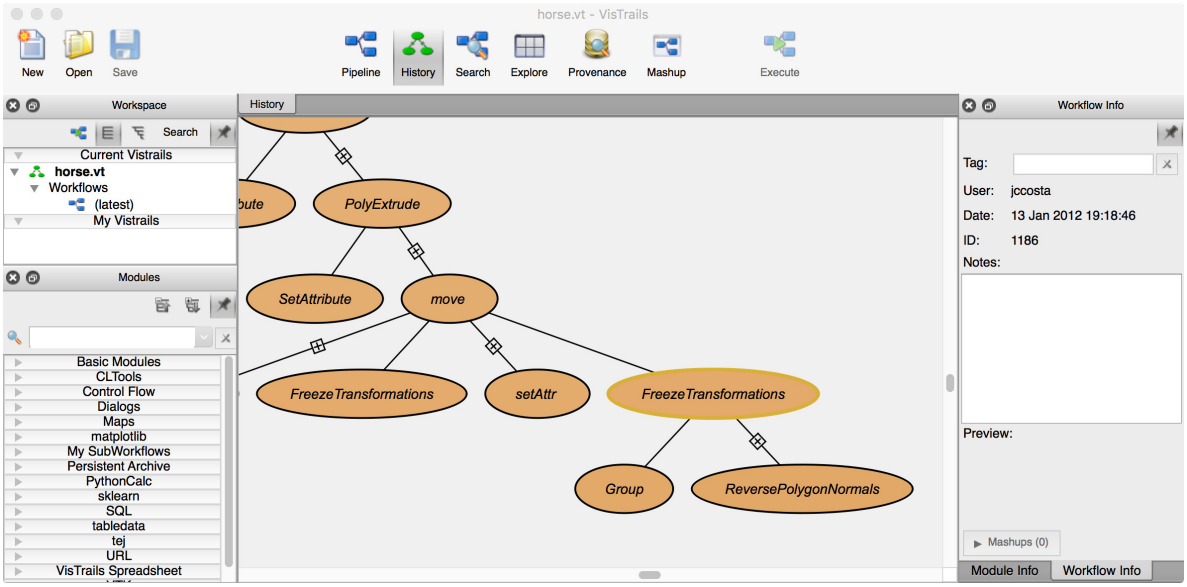
```
1 <action id="184" prevId="183" type_op="local">
2 <annotation id="3214" key="__description__" value="
     performPolyMergeCallback "/>
3 <add id="1116" objectId="1116" what="plugin_data">
4 <plugin_data data="18525935:18539948" id="1116"/></add>
5 <subtree old_id="1117" old_parent="1116"/>
6 </action>
```

Para realizar o Desfazer Localizado, a exibição da subárvore foi necessária em alguns casos. Nestes casos, os nós foram removidos através do Vistrails (N.Y.U., 2005) (Figura 4.2). A mesclagem posterior na árvore principal foi feita tornando o nó removido transparente, i.e., ligando o pai do nó desfeito com seus filhos, mas sem excluir os nós do arquivo *XML*. As operações dependentes das operações desfeitas também são removidas da árvore de controle de versão. Operações dependentes são todas as operações que após a atualização dos parâmetros de suas funções API não encontram elemento da malha correspondente válido sobre o qual atuar.

O mapeamento de vértices e faces é feito através de *geometric hashing* e propagação topológica, quando necessário, para corrigir os locais espaciais de atuação de algumas operações (e atualizar os parâmetros na função API do arquivo *data*). Todas as operações de uma árvore de versões são executadas no Maya (AUTODESK, 2016a) com o plugin já citado para gerar versões modificadas das malhas.

As imagens usadas nesta tese foram geradas usando o Blender (BLENDER, 2016) para demonstrar a flexibilidade do algoritmo que independe do modelador 3D ou de sistema de controle de versões utilizado. Testes de velocidade (Tabela 4.1) foram feitos usando um processador multi-core AMD Opteron(TM) Processor 6276, 64 núcleos (*Hyper-Threading*/hiperprocessamento (*HT*)) com 2.3 10^9 Hertz (GHz) e 8 10^9 Bytes (GB) de *RAM*, um processador Intel(R) Xeon(R) CPU E5640, 8 núcleos com 2.67 GHz, memória *Non-uniform memory access*/acesso não-uniforme da memória (*NUMA*), e um processador Intel(R) Core (TM) Intel (R) i7 4700Socketed

Figura 4.2 – A ACV da construção da malha do cavalo conforme vista pelo Vistrails (N.Y.U., 2005).



Fonte: o autor.

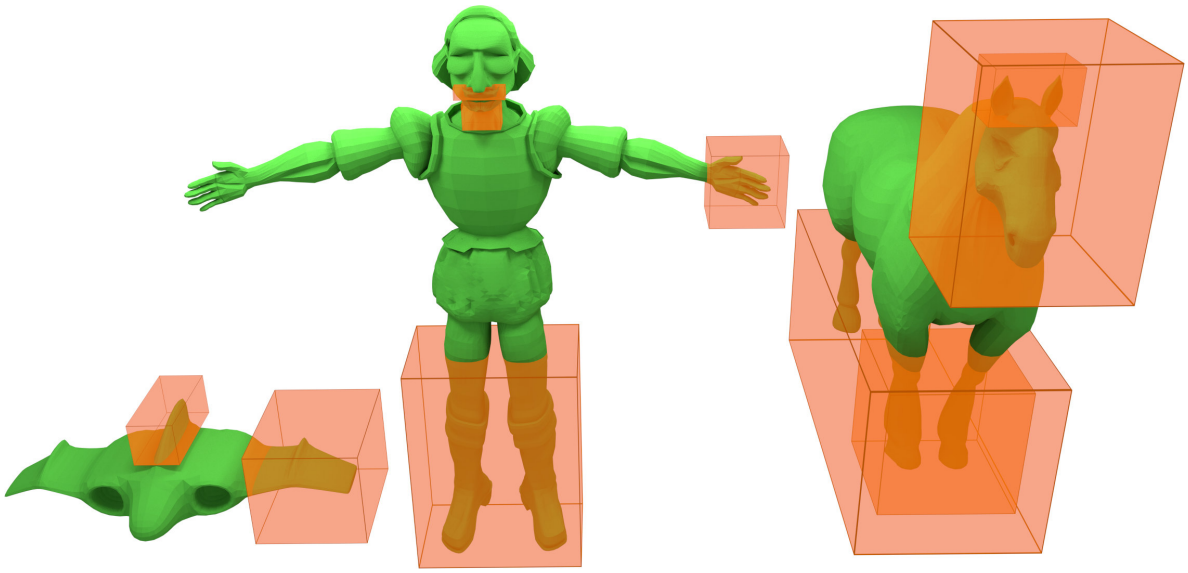
Tabela 4.1 – Análise do algoritmo. O: ACV original; E: árvore enriquecida; S: árvore especializada; e T: subárvore especializada (Figura 3.2).

Malha	Seleção determinada	No. de nós: subárvore / ACV	Tempo		
			$O \to E$	$E \to S$	$S \to T$
Espaçonave	Estabilizador	8 / 92	<1s	<1s	<1s
	Asa Esq.	16 / 92	<1s	<1s	<1s
Cavalo	Orelhas	147 / 1196	<1s	1.4s	<1s
	Cabeça	353 / 1196	<1s	1.7s	<1s
	Patas Front.	104 / 1196	<1s	1.5s	<1s
	Todas Patas	210 / 1196	<1s	1.8s	<1s
Dom Quixote	Bigode	177 / 1648	<1s	5.2s	<1s
	Cavanhaque	144 / 1648	<1s	5.2s	<1s
	Mão Esq.	110 / 1648	<1s	5.4s	<1s
	Pernas	134 / 1648	<1s	7.3s	<1s
Bípede	Mão	340 / 5758	<1s	<1s	<1s
Tubarão	Nariz	396 / 8350	<1s	1.1s	<1s
	Cauda	414 / 8350	<1s	<1s	<1s

Fonte: o autor.

Quad Core (MQ), 8 núcleos (*HT*) com 2.4 GHz e 16 GB de *RAM*. Em seguida, os resultados obtidos para a geração de tutoriais (Seção 4.2) e para o Desfazer Localizado (Seção 4.3) são apresentados.

Figura 4.3 – Versões de 3 malhas obtidas de um Sistema de Controle de Versões (SCV) e seleções de Caixa Envoltória (em laranja) associadas com cada uma delas.



Fonte: o autor.

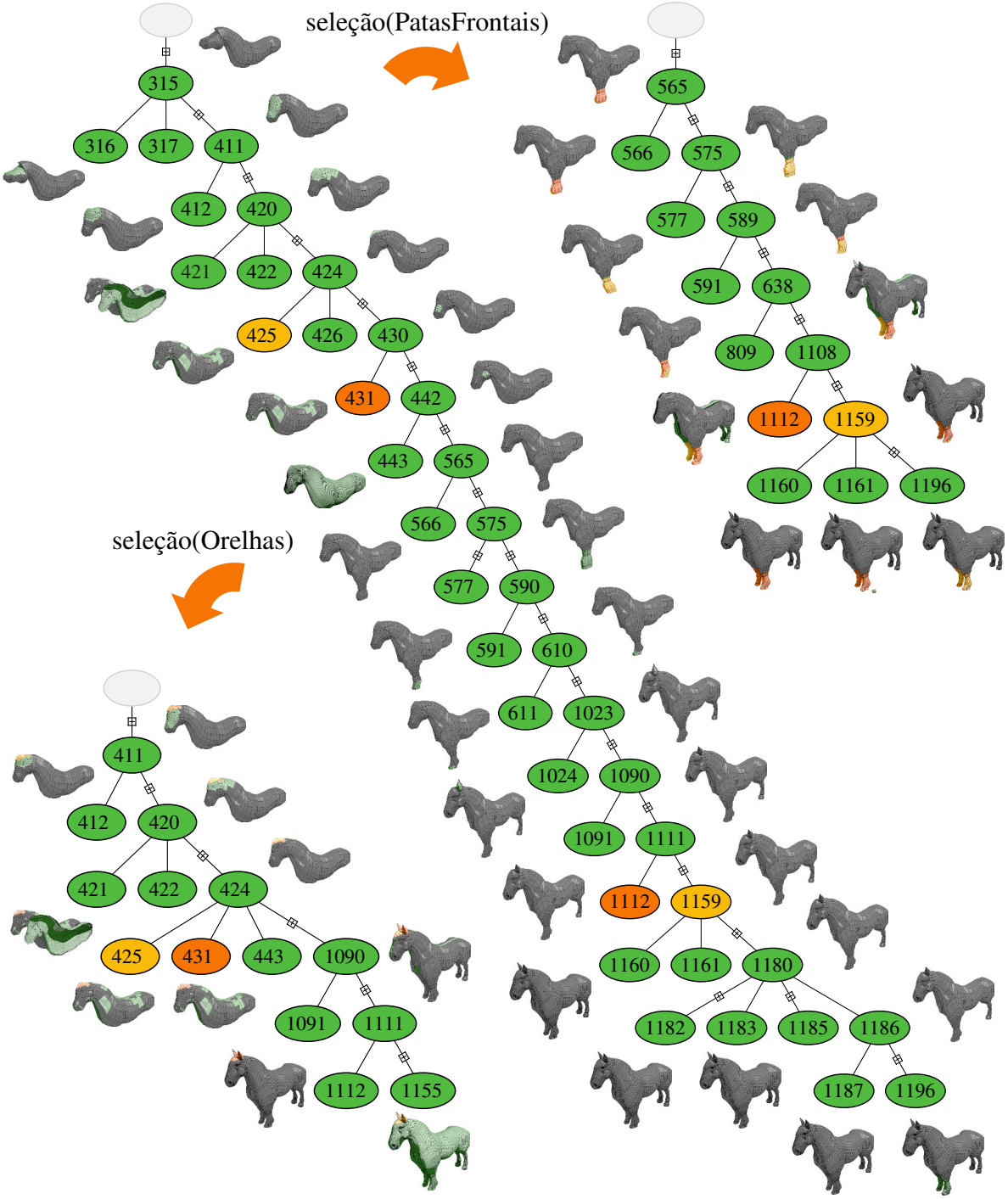
Nota: Através de algumas dessas seleções, subárvores especializadas foram extraídas para criação de tutoriais (Seção 4.2) e para aplicação do Desfazer Localizado (Seção 4.3).

4.2 Exemplos de Tutoriais

Artistas aprendem através da repetição e da observação. Por exemplo, para aprender a desenhar uma mão, um artista pode estudar centenas ou milhares de desenhos de mãos na sua vida. Esse mesmo processo de aprendizagem também é verdade para modelagem 3D. Contudo, quando uma malha está sendo construída, os passos das construções de regiões quaisquer da malha ocorrem de maneira independente, e.g., um artista pode trabalhar alternativamente em duas regiões disjuntas da malha, ou dois artistas podem trabalhar em mais de uma região ao mesmo tempo.

Estas situações determinam que algumas regiões da malha serão totalmente desconhecidas por outro artista que não seja o autor, e mesmo o próprio autor pode não estar seguro de todos os passos que efetuou para uma determinada região ao longo de toda a construção da malha. É neste momento que tutoriais são necessários, os quais podem conter informações textuais relevantes. Tutoriais são guias de aprendizagem sobre um determinado conhecimento, e uma técnica usada para o ensino dos mais variados conteúdos (BAUDRIT, 2003) bastante popular na Internet. Um artista experiente sabe que operações de um modelador estão sendo efetuadas apenas pelo resultado exibido do mesmo modo que um músico experiente reconhece

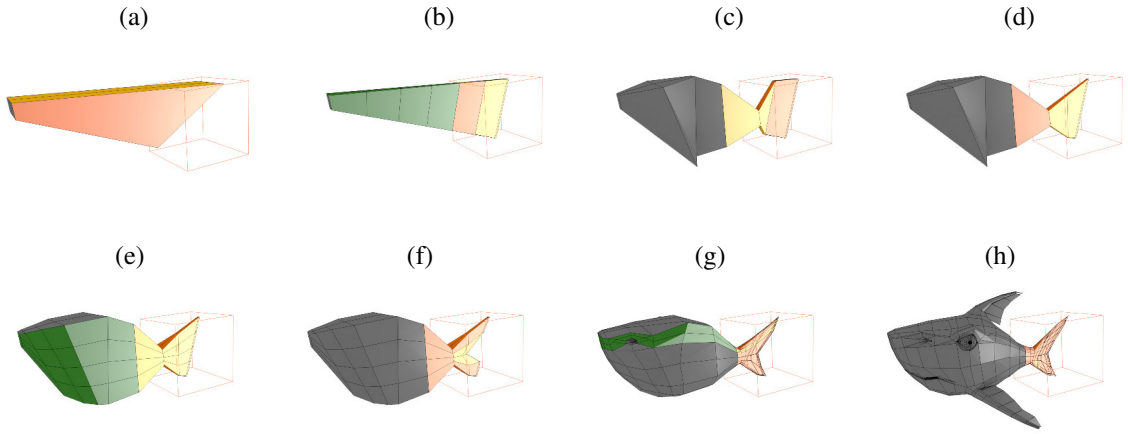
Figura 4.4 – Esta imagem ilustra como tutoriais (Seção 4.2) são feitos: duas árvores especializadas são geradas a partir da árvore principal da malha do cavalo.



Fonte: o autor.

Nota: Nós em amarelo indicam aqueles que preservaram suas relações originais na subárvore, enquanto nós em laranja mostram aqueles que suas relações mudaram. A árvore completa da malha (ao meio) possui 1096 nós, a subárvore especializada criada através da seleção das orelhas possui 147 nós (canto inferior esquerdo) e a subárvore criada através da seleção das patas frontais possui 104 nós (canto superior direito). As malhas usam a mesma paleta de cores da Figura 4.10, e o aumento do tamanho da malha pode ser vista na Figura 4.19.

Figura 4.5 – Uma parte de um tutorial para a construção da cauda do tubarão.



Fonte: o autor.

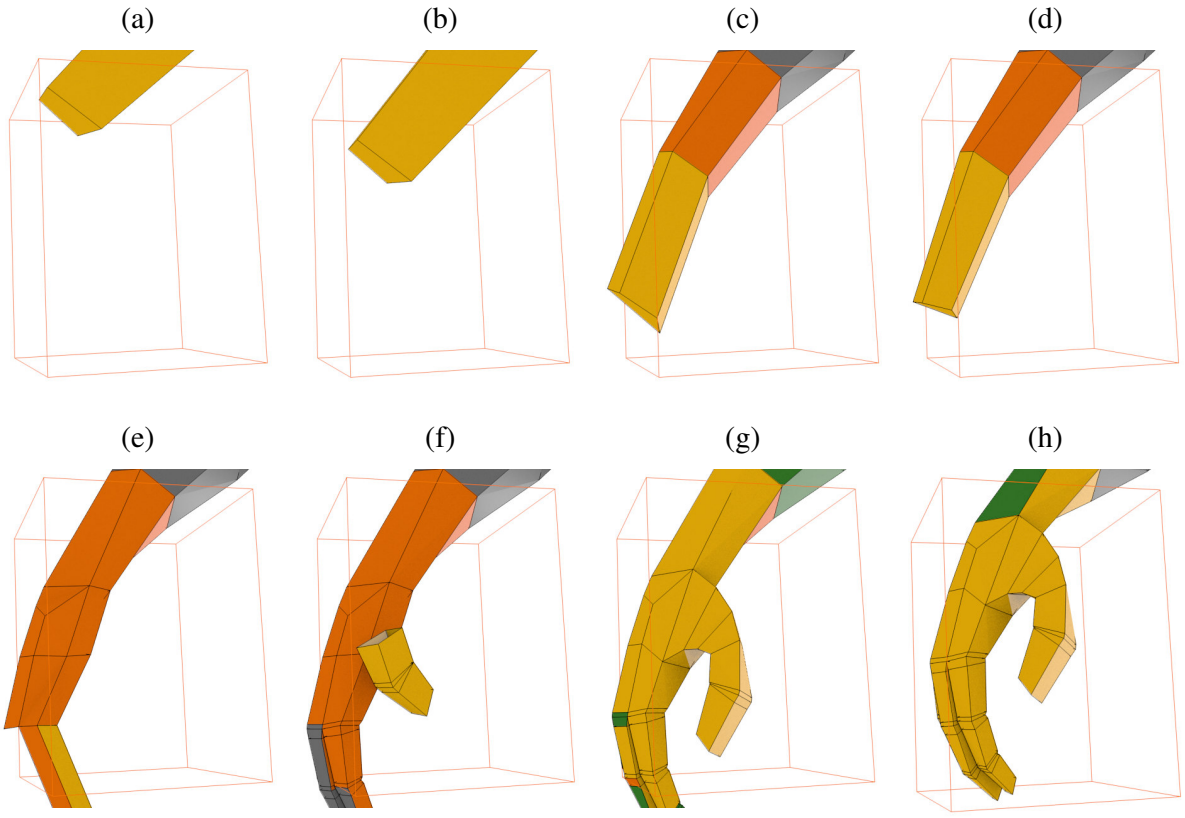
Nota: O Conjunto de Mudanças fora da Caixa Envoltória é exibido em verde, e quando está dentro da Caixa Envoltória é exibido em amarelo. Em laranja as faces que se interceptam com a Caixa Envoltória. Existem 414 operações nesta subárvore gerada. As operações nesta subárvore mostram como um artista pode trabalhar em uma região na malha do modelo várias vezes durante sua construção. As operações de (a) à (d) ocorrem próximo da raiz da ACV, as operações de (e) à (g) no meio da ACV, enquanto a operação (h), que já ocorre próxima do término da malha, é realizada quando o artista efetua toques finais na malha (fim da ACV).

notas sem necessariamente precisar olhar a partitura. Um tutorial visual sem texto pode ser mais rapidamente assimilado do que um que demonstre imagem com texto, o que requer leitura e não pode ser feito em modo de avanço rápido, devido a todas as operações efetuadas (que podem ser muitas, conforme visto na Tabela 4.1). Esta tese exibe tutoriais sem texto que também leva ao aprendizado conforme demonstrado em Grabler *et al.* (2009).

Para construir um tutorial de modelagem com o fim de ensinar como uma região específica de um modelo foi criada para outros artistas (DENNING *et al.*, 2011; GRABLER *et al.*, 2009) é necessário encontrar todas as operações pertinentes a região em ordem, i.e., deve-se encontrar a subárvore de versões especializada na região. Para isto, uma Caixa Envoltória foi usada como uma seleção espacial sobre uma região da malha associada a uma versão determinada (Figura 4.3), e o algoritmo extrai todos os nós cujo o conjunto de interseção entre a Caixa Envoltória e o Conjunto de Mudanças é não vazio respeitando a ordem da árvore principal.

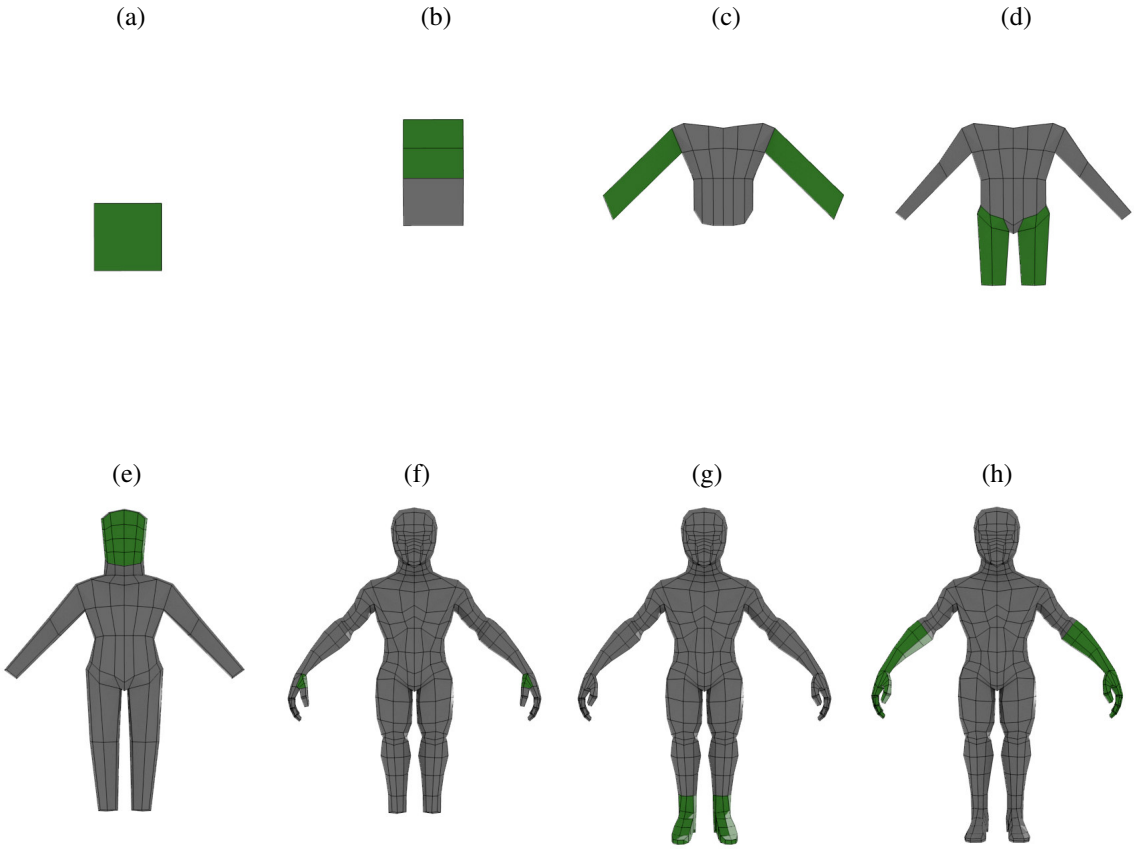
As Sequências de Construção obtidas a partir da subárvore especializada são exemplos de tutoriais para uma região associada, como por exemplo: as patas frontais, assim como as orelhas da malha do cavalo (Figura 4.4); a cauda da malha do Tubarão (Figura 4.5); a mão da malha do Bípede (Figura 4.6); a construção resumida do malha do Bípede (Figura 4.7); as asas da malha da espaçonave (Figura 4.10); e o bigode da malha de Dom Quixote (Figura 4.8). Esses tutoriais são exibidos como vídeos para o artista que pode em qualquer momento pausar para

Figura 4.6 – Uma parte de um tutorial para a construção da mão do Bípede. O Conjunto de Mudanças fora da Caixa Envoltória é exibido em verde, e quando está dentro da Caixa Envoltória é exibido em amarelo. Em laranja as faces que se interceptam com a Caixa Envoltória. Existem 340 operações nesta subárvore gerada. As operações nesta subárvore demonstram como uma caixa envoltória pode conseguir capturar alterações em uma região mesmo com um movimento da região. A Caixa Envoltória foi criada na versão em (h), mas pode-se ver de (a) à (g) que a região da mão teve sua posição alterada durante sua construção. Para solucionar este problema, o artista deve cancelar a operação de Desfazer Localizado ou Tutorial e fazer uma Caixa Envoltória maior, ou uma nova seleção sobre as operações locais que não forem removidas da mão no caso do Desfazer Localizado.



Fonte: o autor.

Figura 4.7 – O tutorial resumido para a construção da malha do Bípede.



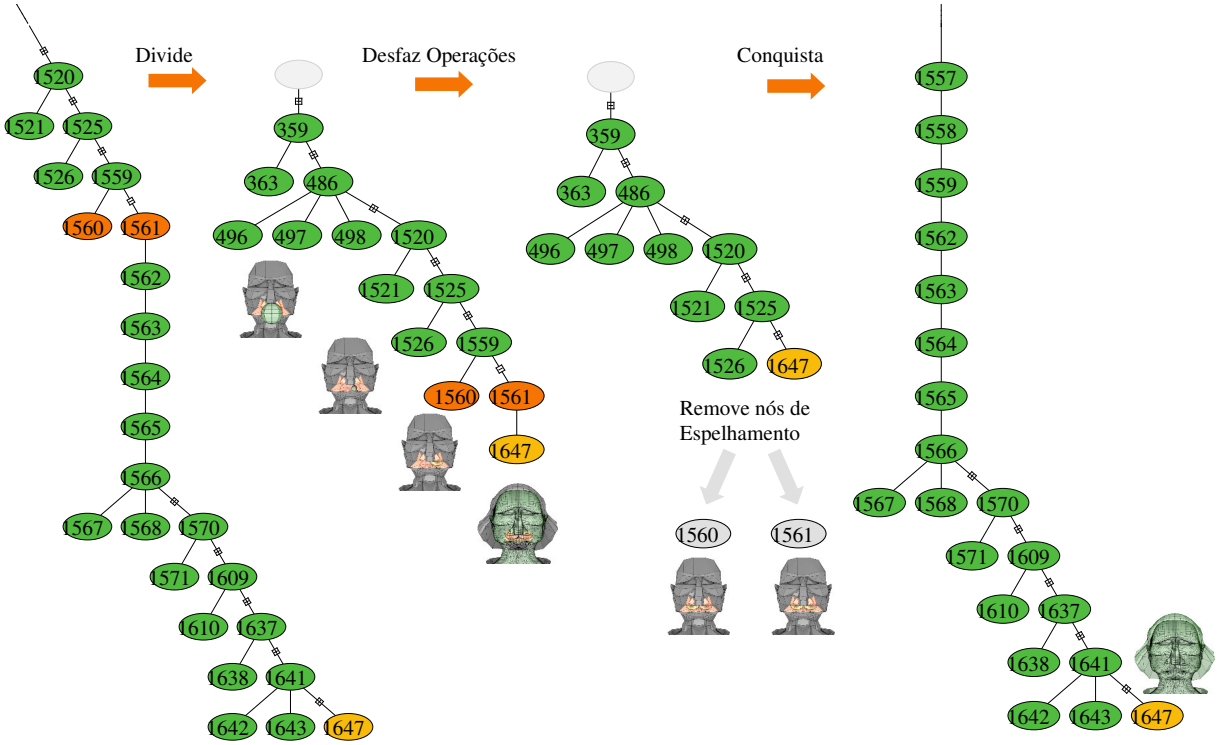
Fonte: o autor.

Nota: O Conjunto de Mudanças é exibido em verde. A ACV possui 5758 operações. As operações nesta ACV mostram uma possível ordem de trabalho de um artista e como isso pode afetar o Desfazer Localizado. Desfazer os pés é possível pela operação de desfazer comum, mas desfazer as mãos sem perder o trabalho nos pés só é possível com Desfazer Localizado.

verificar a operação efetuada, assim como aumentar ou diminuir a velocidade de transmissão dos mesmos (evitando assim agrupar operações conforme DENNING *et al.*, 2011 ou DOBOŠ; STEED, 2012b). Assistir vídeo em $fast - forward$ para aprendizagem também é outra prática muito comum na Internet, tão comum que o HTML5 (HICKSON *et al.*, 2014), o mais novo padrão de arquivo para páginas WEB, já implementa esta opção para todos os vídeos.

A iluminação de ambiente, posição e orientação da câmera foram especificados de acordo com o conjunto de Mudanças dentro da Caixa Envoltória (o foco é a caixa) para melhorar os resultados de saída do algoritmo. As operações de visualização realizadas e armazenadas por um artista nem sempre correspondem a região afetada, ou estão com o foco ajustado para a região de interesse. Isto é, a câmera nem sempre está posicionada e orientada de modo a visualizar os *Changesets* associados (que também podem ocorrer em regiões escuras do modelo). Na Figura

Figura 4.8 – Esta sequência ilustra o Desfazer Localizado (Seção 4.3), explicada sobre o paradigma de dividir-e-conquistar.



Fonte: o autor.

Nota: A subárvore é extraída (177 nós) da árvore principal com 1648 nós (dividir); dois nós são removidos da subárvore que é mesclada novamente na árvore principal (conquistar). O nó 1647 em amarelo mostra uma versão $\mathbb{V}_{q.1}$ e sua versão modificada $\mathbb{V}_{q.1}(\mathbb{V}_{q.2})$ depois que o processo é completado (Figuras 4.9, 4.11, e 4.13 mostram mais exemplos deste processo). As malhas usam a mesma paleta de cores da Figura 4.10, e o aumento do tamanho da malha de Dom Quixote pode ser visto na Figura 4.19.

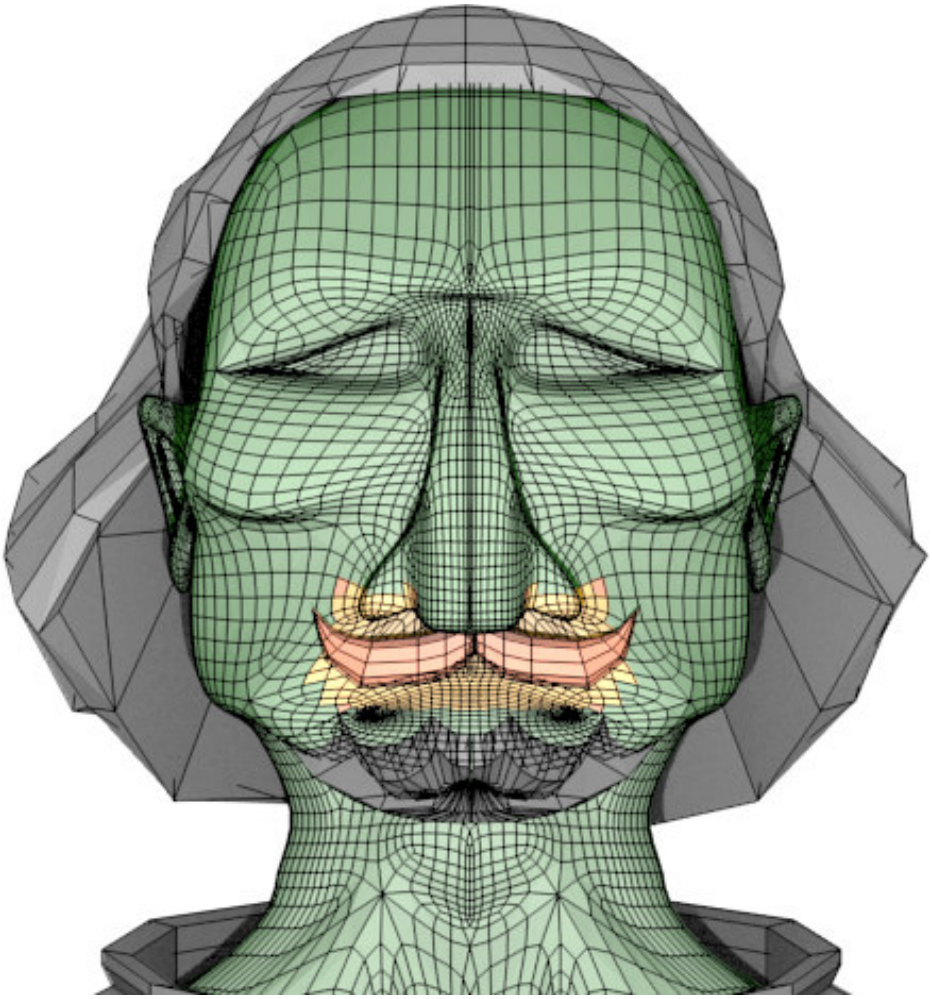
4.11, a câmera foi direcionada para exibir toda a malha, enquanto, na Figura 4.9, a câmera foi direcionada para exibir somente a região selecionada. Em seguida, os resultados do Desfazer Localizado são mostrados (Seção 4.3).

4.3 Exemplos de Desfazer Localizado

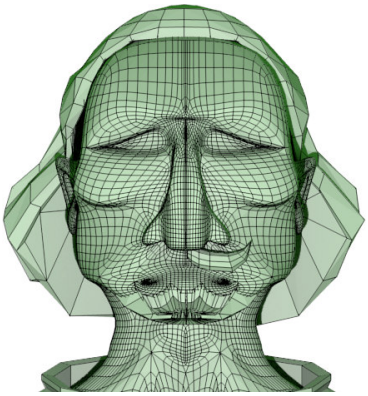
Quando um artista usa um modelador 3D para desfazer um trabalho armazenado em uma ACV, ele possui apenas duas opções atualmente: deletar a região indesejada e recomeçá-la do zero, ou restaurar até uma versão antiga e trabalhar a partir deste momento para gerar uma nova versão em um novo ramo da ACV (*branch*). Contudo, em ambas as opções, ele podou nós da árvore que não necessariamente eram do seu interesse eliminar. O artista pode, claro, tentar mesclar seu novo ramo com o ramo original, porém, dependendo das operações efetuadas na versão, tal trabalho pode ser ainda maior do que refazer toda uma região do zero.

Figura 4.9 – Alterando uma subárvore especializada, foi possível obter o Desfazer Localizado (Seção 4.3): a) Versão original $\mathbb{V}_{q.1}$: a sua paleta de cores é a mesma que a Figura 4.10 usa; b) A versão original $\mathbb{V}_{q.1}$ depois de remover algumas operações no bigode $\mathbb{V}_{q.1}(\mathbb{V}_{q.2})$; c) A versão original $\mathbb{V}_{q.1}$ depois de remover todas as operações no cavanhaque $\mathbb{V}_{q.1}(\mathbb{V}_{q.3})$; d) A versão original $\mathbb{V}_{q.1}$ depois de remover todas as operações no cavanhaque e no cabelo $\mathbb{V}_{q.1}(\mathbb{V}_{q.4})$.

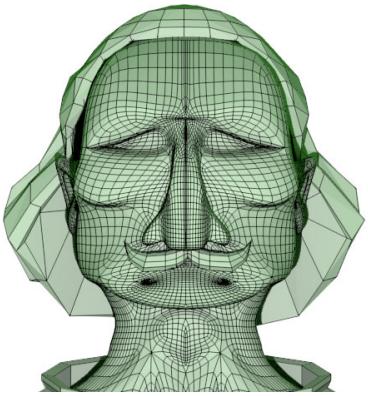
(a) $\mathbb{V}_{q.1}$



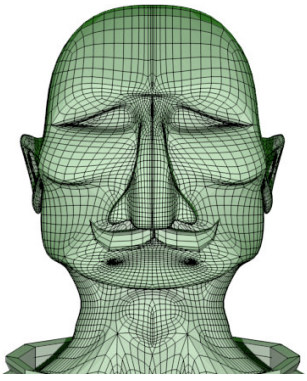
(b) $\mathbb{V}_{q.1}(\mathbb{V}_{q.2})$



(c) $\mathbb{V}_{q.1}(\mathbb{V}_{q.3})$

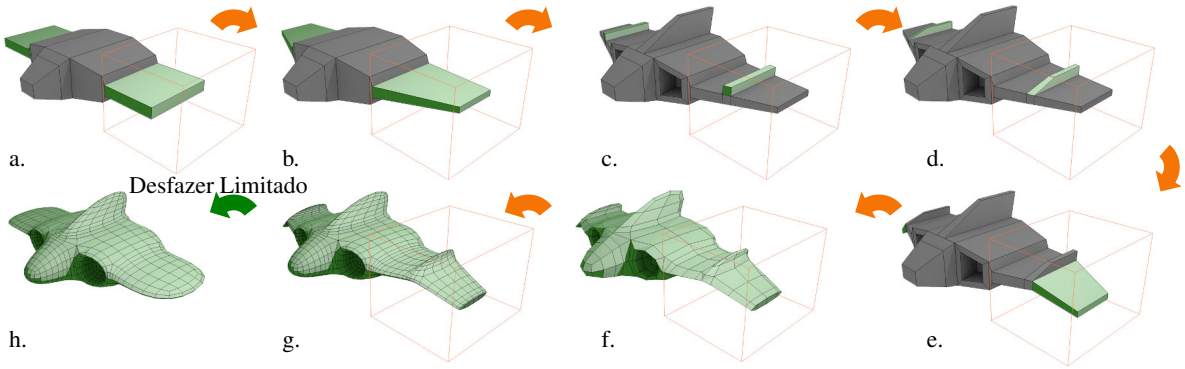


(d) $\mathbb{V}_{q.1}(\mathbb{V}_{q.4})$



Fonte: o autor.

Figura 4.10 – A Sequência de Construção da asa esquerda de uma malha de espaçonave (um tutorial parcial; Subseção 4.2): a) Extrusão de faces; b) Movimento de vértices; c) Extrusão de faces; d) Movimento de vértices; e) Movimento de faces; f-g) Suavização da malha. O Desfazer Localizado (h) foi obtido selecionando-se a asa esquerda (Caixa Envoltória em laranja), e desfazendo-se os passos (e), (d), (c) e (b). Faces em verde estão no Conjunto de Mudanças (i.e., a diferença entre duas malhas na sequência original da árvore principal); e as faces em cinza são inalteradas e fora do Conjunto de Mudanças. Ressalte-se que um simples desfazer não pode alcançar o que foi obtido em (h), pois existem outros passos intermediários entre as operações exibidas que foram removidos (refazer passos da ACV não gera versões novas e (h) não existe na árvore original).



Fonte: o autor.

Através do Desfazer Localizado, desfaz-se operações em determinadas regiões da malha, enquanto outras regiões da malha são preservadas. Isto implica que alguns nós são desfeitos de forma não-linear, diferentemente do que ocorre na operação de desfazer tradicional. Na verdade, pode ser humanamente impossível procurar e remover todos os nós associados com uma região de uma malha independente dos problemas de exibição existentes (NAKAMURA; IGARASHI, 2008). Por exemplo, como a construção de uma malha pode envolver milhares de operações, exibir os Conjuntos de Mudanças ou o nome de uma função API executada pode ser insuficiente para que um usuário entenda a motivação de uma operação conforme detalhado por Nakamura e Igarashi (2008); nem todas as operações envolvem manipulação do modelo; além disso, uma câmera ou luz mal-posicionada pode esconder mudanças que ocorreram na malha completamente.

Para realizar o Desfazer Localizado, uma Caixa Envoltória foi usada como uma seleção espacial sobre uma região da malha (uma versão previamente determinada, como mostra a Figura 4.3) para obter uma subárvore especializada na região. Após a subárvore ser obtida, alguns de seus nós locais foram removidos para criar novas versões na ACV. Por exemplo, alguns detalhes das asas da malha da espaçonave foram removidos, o que criou novas versões

que podem ser vistas na Figura 4.11 sem que nenhum tipo de modelagem ou trabalho artístico fosse necessário.

Pode-se observar no exemplo da espaçonave que, apesar de apenas a asa esquerda ter sido selecionada (Figura 4.3), algumas operações desfeitas afetavam ambas as asas (Figura 4.10), devido a uma conexão de atributos que o Maya permite ser feita (e outros modeladores). Nesse caso, o Desfazer Localizado afetou ambas as asas (Figuras 4.11c e 4.11d). No melhor caso, as operações desfeitas possuem um Conjunto de Mudanças cujo volume ocupado está dentro do volume da seleção. Este evento não é raro de ocorrer, e depende do tamanho da Caixa Envoltória e das operações utilizadas pelo artistas: um exemplo do melhor caso possível é exibido na Sequência de Construção das patas frontais da malha do cavalo (Figura 4.4) que também foram usadas no exemplo de desfazer na Figura 4.12.

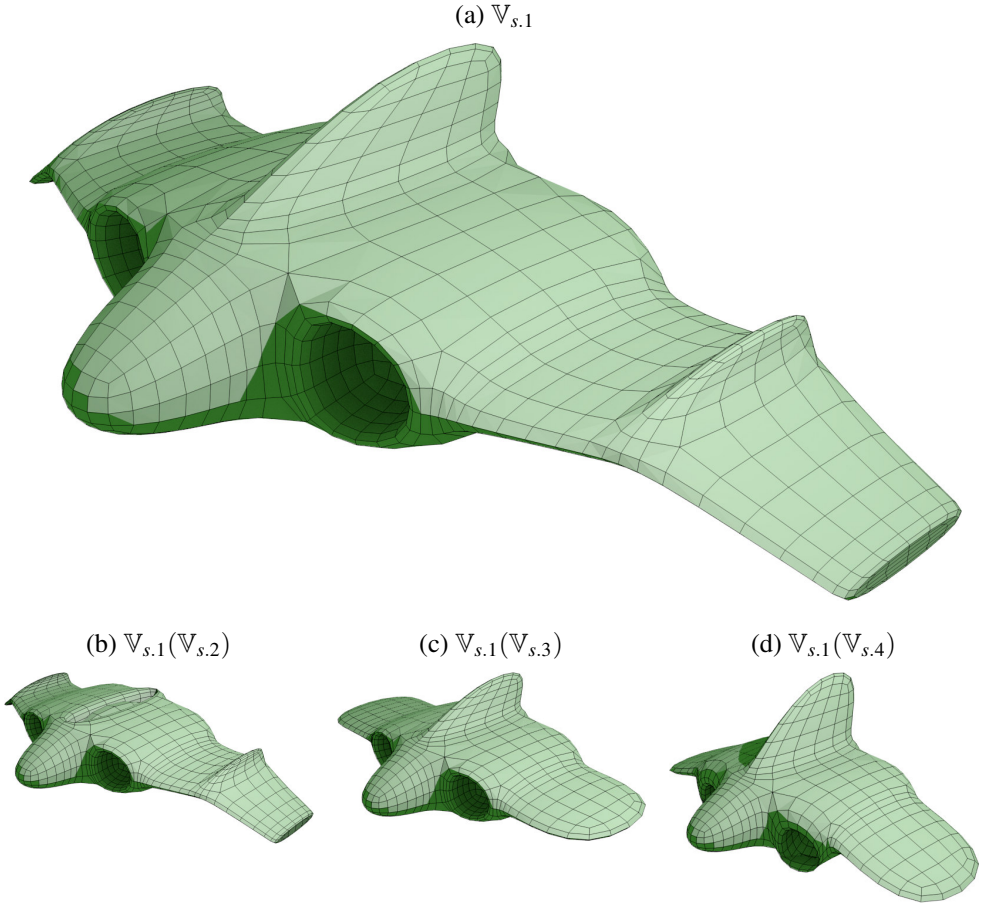
O exemplo na Figura 4.12 também demonstra como o Desfazer Localizado pode ser equivalente a uma operação de deleção conforme o caso. As patas são criadas por operações sequênciais de extrusão. desfazer tais operações mesmo com a atualização de parâmetros não desfez o buraco do local em que as patas deveriam estar.

Um exemplo de caso desfavorável do algoritmo é quando o Conjunto de Mudanças afeta regiões além do volume selecionado podendo gerar algumas deformações ou elementos indesejáveis. Alguns desses casos podem ser vistos na malha de Dom Quixote, pois as regiões foram desfeitas em casos desfavoráveis: bigode (Figuras 4.8 e 4.9b), cavanhaque (Figura 4.9c), e cabelo (Figura 4.9d). A operação de Desfazer Localizado também é aplicada em um caso desfavorável sobre as orelhas da malha do cavalo (Figure 4.13), gerando algumas deformações na cabeça da malha do cavalo. Perceba que mesmo em casos não-favoráveis, o algoritmo consegue gerar resultados razoáveis.

Em todos os exemplos, o Desfazer Localizado foi aplicado sobre a versão original removendo nós da subárvore especializada. Todas as versões afetadas dentro de uma ACV dependem de quais operações locais foram desfeitas, sendo possível existir caminhos na árvore que não são afetados pelo Desfazer Localizado (um dos pontos chaves para a assegurar a velocidade do algoritmo). Os nós removidos tornam-se invisíveis na árvore principal, mas ainda existem de forma que é possível restaurar o modelo para a versão original contida na ACV.

Como último exemplo ilustrativo da capacidade do Desfazer Localizado usa-se uma malha de cruz 2D da Figura 4.14. Essa malha é criada com 5 operações do modelador: uma inserção de face, e quatro extrusões de arestas. Na última versão da malha $\mathbb{V}_5$, faz-se uma seleção

Figura 4.11 – Alterando uma subárvore especializada, foi possível obter o Desfazer Localizado (Seção 4.3): a) Versão original $\mathbb{V}_{s.1}$; b) A versão original $\mathbb{V}_{s.1}$ depois de remover todas as operações no estabilizador vertical $\mathbb{V}_{s.1}(\mathbb{V}_{s.2})$; c) A versão original $\mathbb{V}_{s.1}$ depois de remover algumas operações nas asas $\mathbb{V}_{s.1}(\mathbb{V}_{s.3})$; d) A versão original $\mathbb{V}_{s.1}$ depois de remover algumas operações nas asas e algumas operações no estabilizador vertical $\mathbb{V}_{s.1}(\mathbb{V}_{s.4})$.

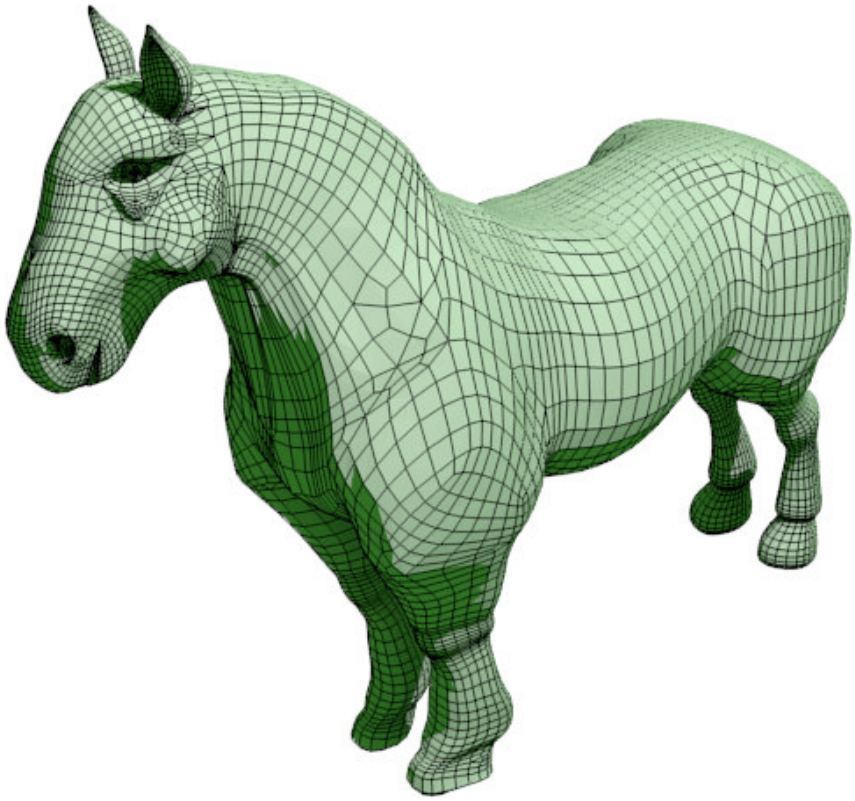


Fonte: o autor.

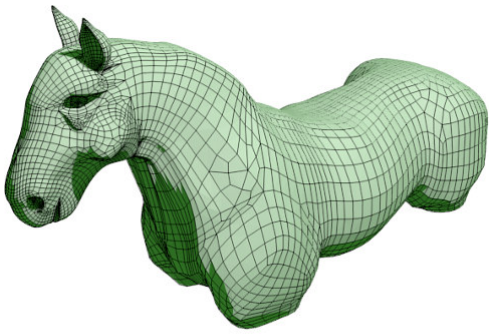
no centro da malha da cruz. Essa seleção é propagada por todos os nós da árvore como pode ser visto na Figura 4.15. Somente um nó está na subárvore obtida, o nó N_2, que é um nó local. Devido às referências criadas na fase de cálculo do Conjunto de Mudanças, é possível remover o nó N_2 e mapear as outras operações para os vértices de origem da versão $\mathbb{V}_2$ da malha conforme pode ser visto na Figura 4.16. Como algumas arestas são mapeadas por autorreferência a vértices, operações de extrusão em arestas tornam-se extrusão em vértices após o Desfazer Localizado.

Figura 4.12 – Mudando a subárvore especializada, é possível obter o Desfazer Localizado (Seção 4.3) que em alguns casos pode ser equivalente a uma operação de deleção. Neste caso a subárvore possui 210 nós e apenas 3 deles são globais (624, 626, 1155; 2 operações de translação e 1 de suavização): a) Versão original $\mathbb{V}_{h.1}$; b) Visão lateral da versão original $\mathbb{V}_{h.1}$ depois de remover todas as operaçõs nas pernas $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$; c) Visão inferior da versão original $\mathbb{V}_{h.1}$ depois de remover todas as operaçõs nas pernas $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$.

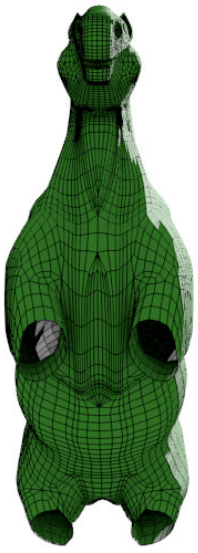
(a) $\mathbb{V}_{h.1}$



(b) $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$

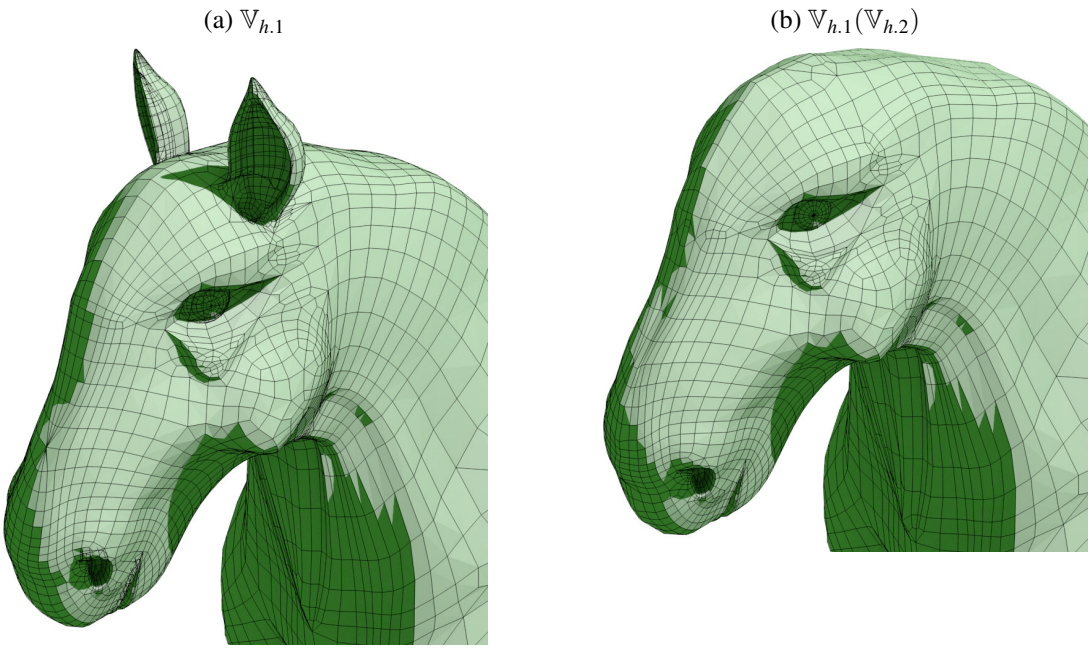


(c) $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$



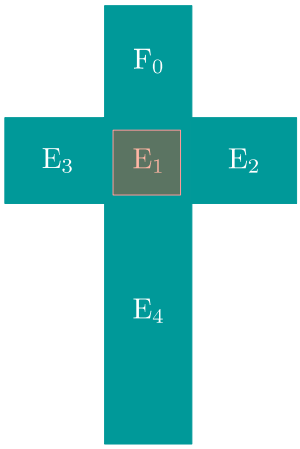
Fonte: o autor.

Figura 4.13 – Mudando a subárvore especializada, é possível obter o Desfazer Localizado (Seção 4.3): a) Versão original $\mathbb{V}_{h.1}$; b) A versão original $\mathbb{V}_{h.1}$ depois de remover todas as operaçõs nas orelhas $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$.



Fonte: o autor.

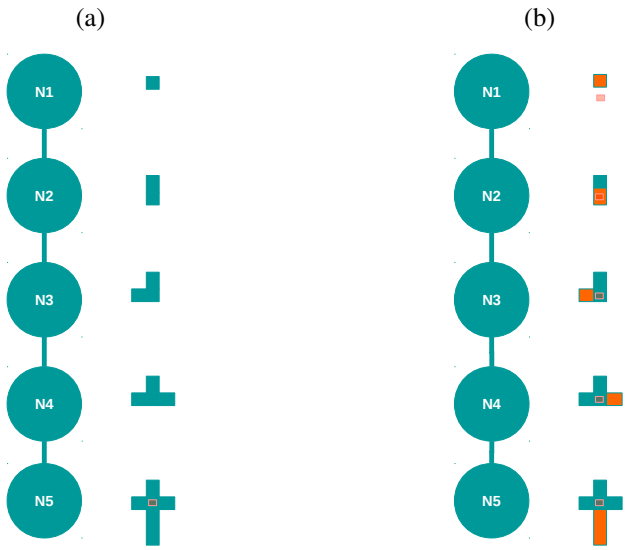
Figura 4.14 – Uma simples ACV em que apenas 5 operações foram efetuadas.



Fonte: o autor.

Nota: F_0: inserção de face, E_1: extrusão de aresta, E_2: extrusão de aresta, E_3: extrusão de aresta, e E_4: extrusão de aresta. Pode-se observar a versão $\mathbb{V}_5$ da malha da cruz e uma seleção por Caixa Envoltória é feita no seu meio para remover o centro da cruz por Desfazer Localizado.

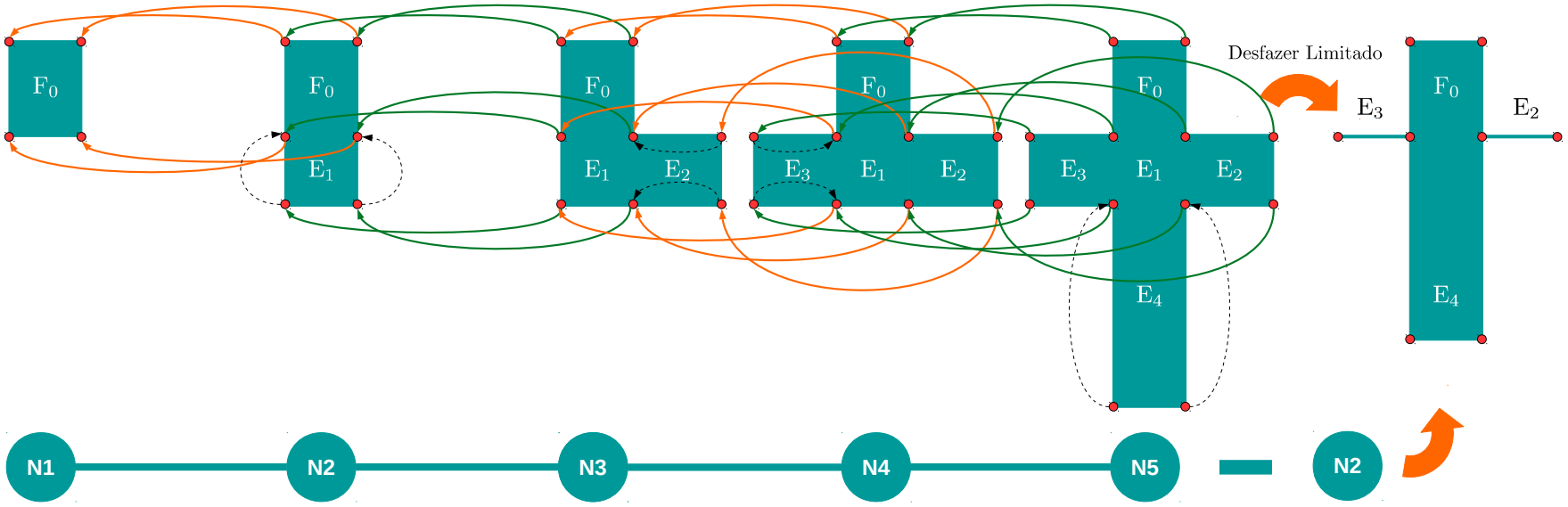
Figura 4.15 – Uma simples ACV em que apenas 5 operações foram efetuadas.



Fonte: o autor.

Nota: N_1: inserção de face, N_2: extrusão de aresta, N_3: extrusão de aresta, N_4: extrusão de aresta, e N_5: extrusão de aresta. Pode-se observar a construção da versão $\mathbb{V}_5$ da malha da cruz com uma seleção por Caixa Envoltória em N_5 (a); e o resultado da propagação da Caixa Envoltória a partir do nó N_5 da árvore, onde somente o segundo nó N_2 coincide com a seleção (b). Os conjuntos de mudança estão pintados de laranja.

Figura 4.16 – Nesta figura, pode-se obervar a construção de uma malha de cruz 2D.



Fonte: o autor.

Nota: Cada operação efetuada corresponde a um nó da ACV, que contém 5 operações: F_0, inserção de face; E_1, extrusão de aresta; E_2, extrusão de aresta; E_3, extrusão de aresta; e E_4, extrusão de aresta. Após a remoção do segundo nó por Desfazer Localizado obtém-se uma nova malha da cruz. As setas em vermelho e verde indicam referências entre malhas de diferentes versões, e setas pretas pontilhadas indicam casos de autorreferência. Conforme a operação E_1 é removida da ACV, o que era uma extrusão de arestas torna-se uma extrusão de vértices devido ao mapeamento de arestas para vértices, e somente a face obtida pela operação E_4 é preservada.

4.4 Análise da eficiência do mapeamento por tabelas *hash*

Nesta seção é feita uma análise da eficiência do mapeamento de vértices por tabelas *hash*. O Fator de Uso f_u de uma tabela *hash* é uma das métricas de análise mais usada para este tipo de estrutura de dados (SEDGEWICK; WAYNE, 2011), e é expresso por

$$f_u = \frac{m}{n}, \tag{4.1}$$

em que m é o número de registros que são armazenados na tabela *hash*, e n é o número de espaços disponíveis na tabela. Esse número deve ser tal que $f_u \in [0,1]$ para garantir acesso aos elementos armazenados em tempo linear. Conforme pode-se observar na Tabela 4.2[2], o algoritmo mantém o fator estabelecido e garante o acesso em tempo linear aos dados. Se f_u se aproxima de 1 o número de colisões tende a ser maior (mais processamento é usado), e se f_u se aproxima de 0, existem muitos espaços vazios na tabela *hash* (mais memória é usada).

Na Tabela 4.2, a coluna da taxa de ocupação média fornece a soma dos valores máximos de vértices dentro de cada bucket para cada tabela divididos pela quantidade de vértices no nó correspondente e em seguida dividido pelo número de nós ao longo de toda a ACV. Como os valores máximos são usados como referência, é possível deduzir que a taxa de ocupação real é ainda menor que o valor apresentado em tempo de execução. Já as colunas sob Quantidade de Vértices nos exibe a quantidade máxima de vértices encontrada em um *bucket* qualquer ao longo de todos os nós da ACV ao lado da quantidade de vértices que existe para a versão da malha correspondente.

O pior caso da Tabela 4.2, os maiores valores, ocorrem com malhas pequenas como a da Espaçonave, que não pode usufruir muito do uso das tabelas *hash*, mas, mesmo neste caso, a ocupação média foi consideravelmente baixa, 2.2%, com 3 vértices no maior *bucket*. De posse destes dados, pode-se concluir que o algoritmo tal qual implementado, usando de divisão hierárquica do espaço aliada as chaves *hash*, consegue um grande ganho de eficiência. Deste modo, a comparação entre elementos da malha ocorre certamente em tempo linear, e este fato se demonstra na velocidade obtida pelo algoritmo que é discutida na próxima seção (Seção 4.5).

[2] Os modelos do Helmo, Hidrante, Tubarão e Bípede estão presentes na tese de Denning e foram todos executados sobre a técnica aqui implementada para coleta de dados, embora nem todos tenham suas figuras exibidas.

Tabela 4.2 – Análise do Fator de Uso das Tabelas *Hash*.

Malha	Fator de Uso ($load\ factor$)	Taxa de ocupação	Quantidade de Vértices	
			Bucket	Malha
Espaçonave	0.54	2.2%	3	1860
Cavalo	0.97	0.2%	16	10896
Dom Quixote	0.96	0.3%	23	7168
Helmo	0.94	0.7%	8	128
Hidrante	0.98	0.1%	20	31852
Tubarão	0.95	0.3%	7	384
Bípede	0.94	0.6%	8	640

Fonte: o autor.

4.5 Análise da eficiência obtida com a paralelizaçcão do algoritmo

Para efetuar os testes de aumento do Fator de Aceleração f_a foi usado um Core i7 – 4700 MQ com 4 núcleos usando *Hyper-threading* e um Xeon E5640 de 8 núcleos com memória *NUMA*. O cálculo do fator segue a lei de Amdahl (AMDAHL, 1967):

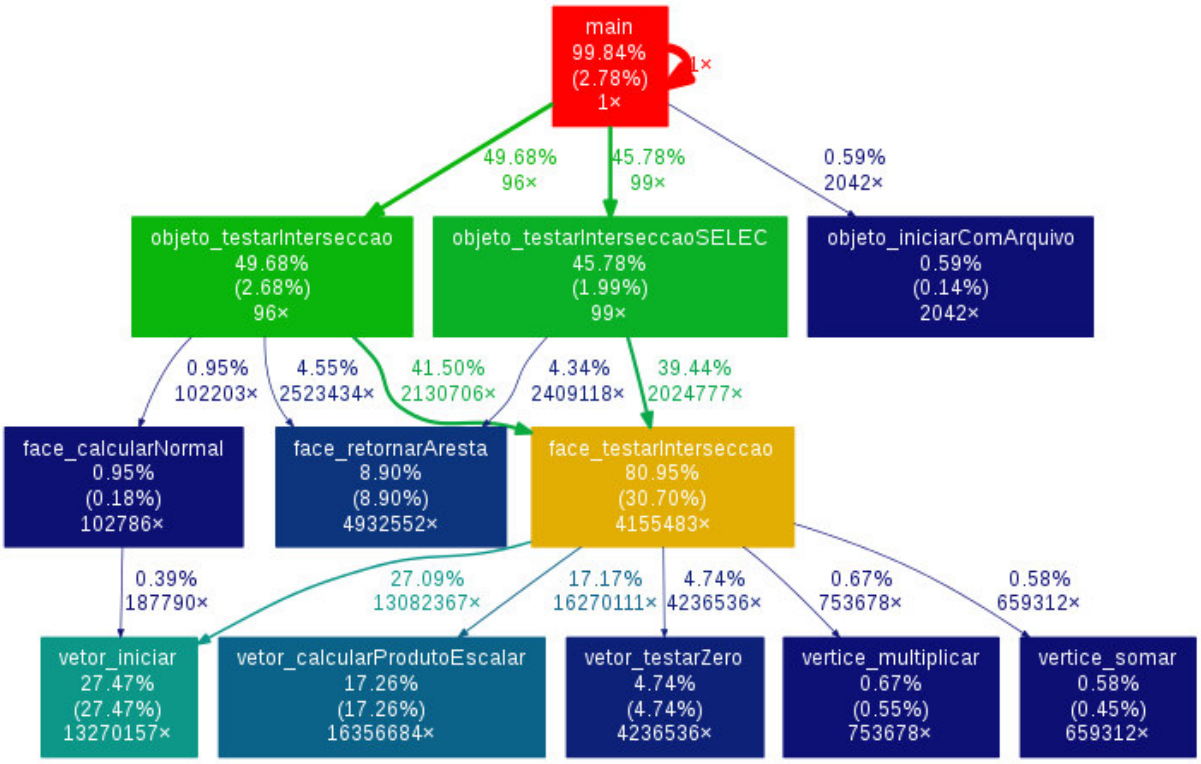
$$f_a = \frac{p}{q}, \tag{4.2}$$

tal que p é o tempo de execução do programa de forma serial (em um processador), e q é o tempo de execução do programa de forma paralela. Conforme pode-se observar nos gráficos da Figura 4.18, a velocidade do algoritmo executado de forma serial aumenta conforme mais núcleos são utilizados, e chega-se próximo da velocidade ideal esperada principalmente na malha do Cavalo (Figura 4.18b). É interessante observar que o tamanho da Malha do Cavalo é intermediário entre a malha da Espaçonave e de Dom Quixote. Isto implica que o tamanho total das malhas não é um fator de grande influência na velocidade.

Na Figura 4.17, pode-se observar o gráfico de chamadas de funções utilizadas pela técnica implementada quando esta é executada de forma serial. A maior parte da computação está concentrada nos testes de interseção conforme esperado para um algoritmo de Geometria Computacional (BADAWY; AREF, 1999).

É interessante notar a relação entre as Figuras 4.17 e 4.18. Se o tamanho total dos elementos das malhas não influencia diretamente o fator de aceleração, mas a maior parte do tempo de execução do algoritmo é dedicada aos testes de interseção, pode-se inferir que o fator determinante na velocidade é, na verdade, o tamanho dos Conjuntos de Mudanças que se interceptam com a Caixa Envoltória, já que é o local em que os testes de interseção ocorrem.

Figura 4.17 – Nesta figura é exibido o gráfico de chamadas para a técnica utilizada através do programa de *profile* gprof.



Fonte: o autor.

Nota: As cores do vermelho até o azul, através da gradação do espectro visível de cores (NASA, 2016), indicam que processos concentram o maior tempo de execução serial das funções do algoritmo, além de indicar o número de chamadas efetuadas para cada função. Cada nó pai da árvore indica como o tempo de determinada função da técnica foi distribuído entre seus nós filhos. Neste caso, o algoritmo foi executado de uma versão da malha do Cavalo com a seleção de todas as patas.

4.6 Comparação com Técnica no Estado da Arte

Nesta seção compara-se a técnica proposta com uma concorrente do estado da arte para sistemas de controle de versão para malhas. A técnica proposta é comparada com a técnica 3DFlow contida na tese de Denning (DENNING, 2014), e publicada com melhorias posteriormente (DENNING *et al.*, 2015).

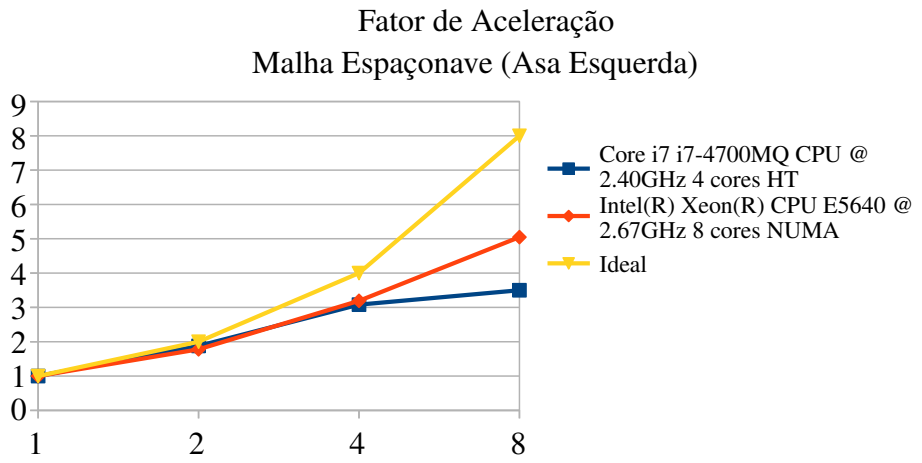
Para efetuar a comparação com 3D Flow foi usado um Core i7 2.40 Ghz com 16 MB *RAM* enquanto o computador usado por Denning foir um Core i7 2.93 Ghz com 16 MB de *RAM* e uma placa de vídeo ATI Radeon HD 5750. O resultado obtido por esta tese não se utiliza de nenhum recurso da placa de vídeo ao contrário do trabalho de Denning.

Na Tabela 4.3[3] estão presentes alguns dos modelos tridimensionais exibidos na tese

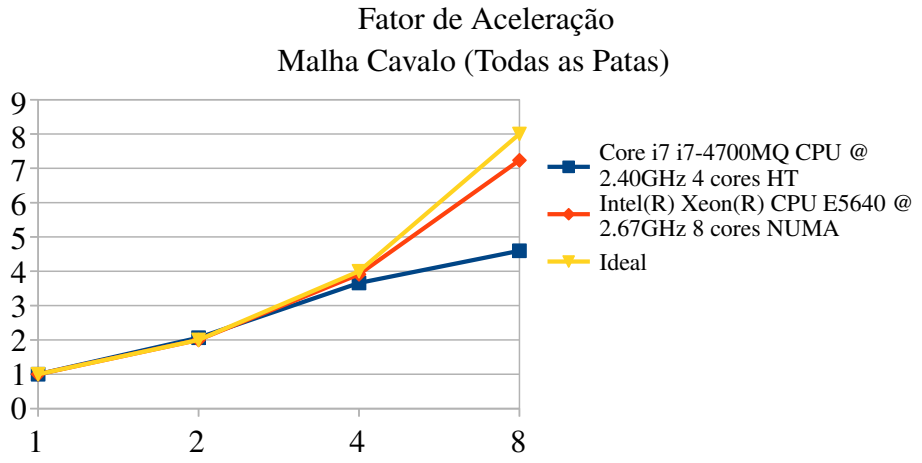
[3] Os modelos aqui descritos estão todos exibidos na tese de Denning e foram todos executados sobre a técnica aqui implementada para coleta de dados, embora nem todos tenham suas figuras exibidas.

Figura 4.18 – O fato de aceleração (*Speed-up*) é um dos principais indicadores da eficiência de um algoritmo ao ser executado em paralelo.

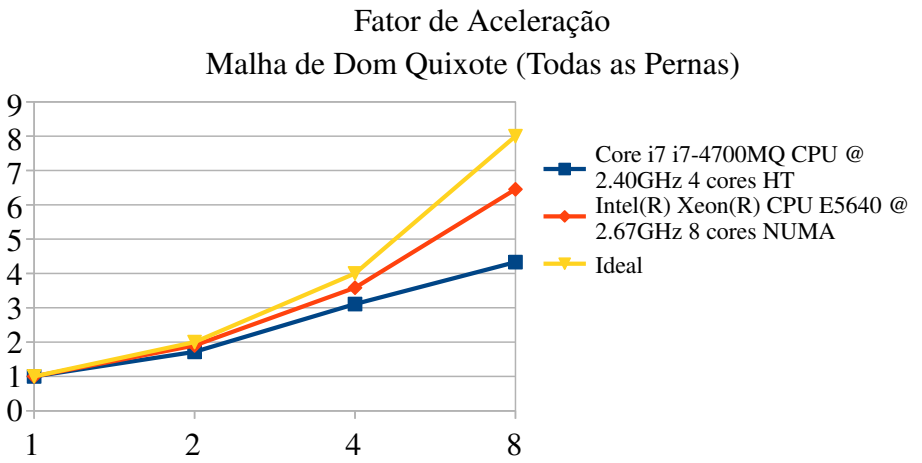
(a)



(b)



(c)



Fonte: o autor.

Nota: O cálculo do fator segue a lei de Amdahl (AMDAHL, 1967). Nos gráficos acima são exibidos a execução do algoritmo em duas máquinas topo de linha atualmente, um Xeon para servidores com 8 núcleos e um core i7 de quarta geração com 4 núcleos.

Tabela 4.3 – Comparação da técnica exibida nesta tese e o algoritmo de Denning.

Malhas	Conjuntos de Mudanças	Tempo	
		tese	3DFlow
Helmo	1321	1.7s	5s
Hidrante	691	1.25s	4s
Robô	1810	8s	15s
Tubarão	1457	1.1s	6s
Bípede	1267	0.45s	5s

Fonte: o autor.

de Denning para comparação de velocidade; é possível observar uma grande vantagem que esta técnica oferece ao ser comparada com a de Denning. Essa comparação se limita a geração de tutoriais, visto que o autor não realiza qualquer tipo de desfazer especialista em seu trabalho, e que o formato de arquivo fornecido pelo autor não é compatível com a implementação do algoritmo para o Desfazer Localizado (que depende do modelador utilizado).

A análise efetuada sugere que usar distância de edições (*Mesh-editing*), mesmo com fatores topológicos e geométricos, para o cálculo de Conjunto de Mudanças não parece ser o caminho mais indicado a se perseguir em sistemas de controle de versões para malhas, e que tabelas de dispersão (*hash*) oferecem uma vantagem significativa a ser considerada, pois podem ser associadas a hierarquia de subdivisão espacial (BRUNET; NAVAZO, 1990), índice de bit para mapeamento (CHAN; IOANNIDIS, 1998), e outras técnicas determinísticas, ao contrário de técnicas heurísticas cuja associação pode gerar mais problemas do que benefícios para o mapeamento de elementos de uma malha, que exige precisão.

Essa vantagem ocorre além dos fatores de velocidade (Tabela 4.3) em precisão, reduzindo número de erros de referência para zero. Conforme é descrito na técnica da tese Denning podem acontecer referências erradas na associação de elementos da malha (Vértices, Arestas e Faces) e o autor os reconhece em seu trabalho.

4.7 Discussão

Nesta seção são discutidas as limitações atuais da técnica. A subárvore especializada pode conter operações onde o Conjunto de Mudanças ocupa um espaço além do volume da seleção. Por exemplo, uma subárvore especializada foi gerada selecionando o espaço em volta do bigode com uma Caixa Envoltória (Figura 4.3). Contudo, já que essa seleção se intercepta com parte da face (Figura 4.9a), algumas operações que não são relacionadas com o bigode estão presentes na subárvore (note que essas regiões estão dentro do volume da seleção). Isto implica

que a remoção de todas as operações locais da subárvore do bigode, não apenas removeria o bigode, mas parte das operações realizadas no nariz e face, deformando o modelo. Apesar disso, este é um problema específico da forma que a seleção que foi usada, e ela pode ser melhorada para conter apenas o bigode.

A inclusão de algumas operações que não afetam a região de interesse não afeta o tamanho final das subárvores especializadas, que permanece menor que toda a árvore de versões – já que a subárvore é um membro do conjunto das partes da ACV, raramente a subárvore é igual ao conjunto. Para que este caso ocorra, é necessário que a Caixa Envoltória envolva toda uma versão ou que todos os conjuntos de Mudanças afetem todos os elementos de todas as versões (mesmo assumindo tamanhos aleatórios de subárvores obtidas por Caixas Envoltórias, a probabilidade de que todos os nós estejam na subárvore é de $1/2^n$, em que n é o número de nós na árvore). Para os casos observados nesta tese, a maior subárvore obtida não chega a conter nem 30% dos nós da árvore completa (Tabela 4.1) contendo em média 10% dos nós da ACV e, além disso, as subárvores são melhores para realizar interação e permitir a visualização do usuário.

A ordem das operações realizadas também é um fato crucial do Desfazer Localizado, principalmente se deseja-se remover todas as operações locais de uma região, visto que operações globais não podem ser removidas. A inserção de um componente desconexo da malha existente gera uma operação global que não pode ser removida. Por exemplo, se um braço foi criado a partir de uma mão, desfazer todas as operações locais que afetam a mão que foi inserida por uma operação global pode ocasionar na perda do braço completamente. O artista, em tal situação, ainda pode deletar a mão, mas o Desfazer Localizado não o ajuda nesta situação; apenas na situação reversa e de **crescimento natural**, o Desfazer Localizado pode ajudar os artistas (caso a intenção seja de remover todas as operações locais de uma região). Crescimento natural significa que a criação de um modelo ocorre conforme na natureza, ou seja o núcleo de um modelo é criado antes de suas extremidades. Apesar disso, o Desfazer Localizado, mesmo no caso anterior, pode ajudar o artistas a desfazer parcialmente as operações do modelo da mão sem afetar o restante do braço.

O Desfazer Localizado, conforme foi explicado, pode afetar mais do que a região desejada se o Conjunto de Mudanças estender-se para além do volume obtido pela Caixa Envoltória. Por exemplo, a asa esquerda da malha da espaçonave foi selecionada (Figura 4.10), mas algumas operações afetavam ambas as asas (seus atributos estavam conectados). Portanto, a remoção das operações da asa esquerda afetam a asa direita (Figura 4.11). Para solucionar

este caso, seria necessário quebrar uma operação em suboperações menores de acordo com os *Changesets* obtidos, que é uma sugestão para trabalhos futuros.

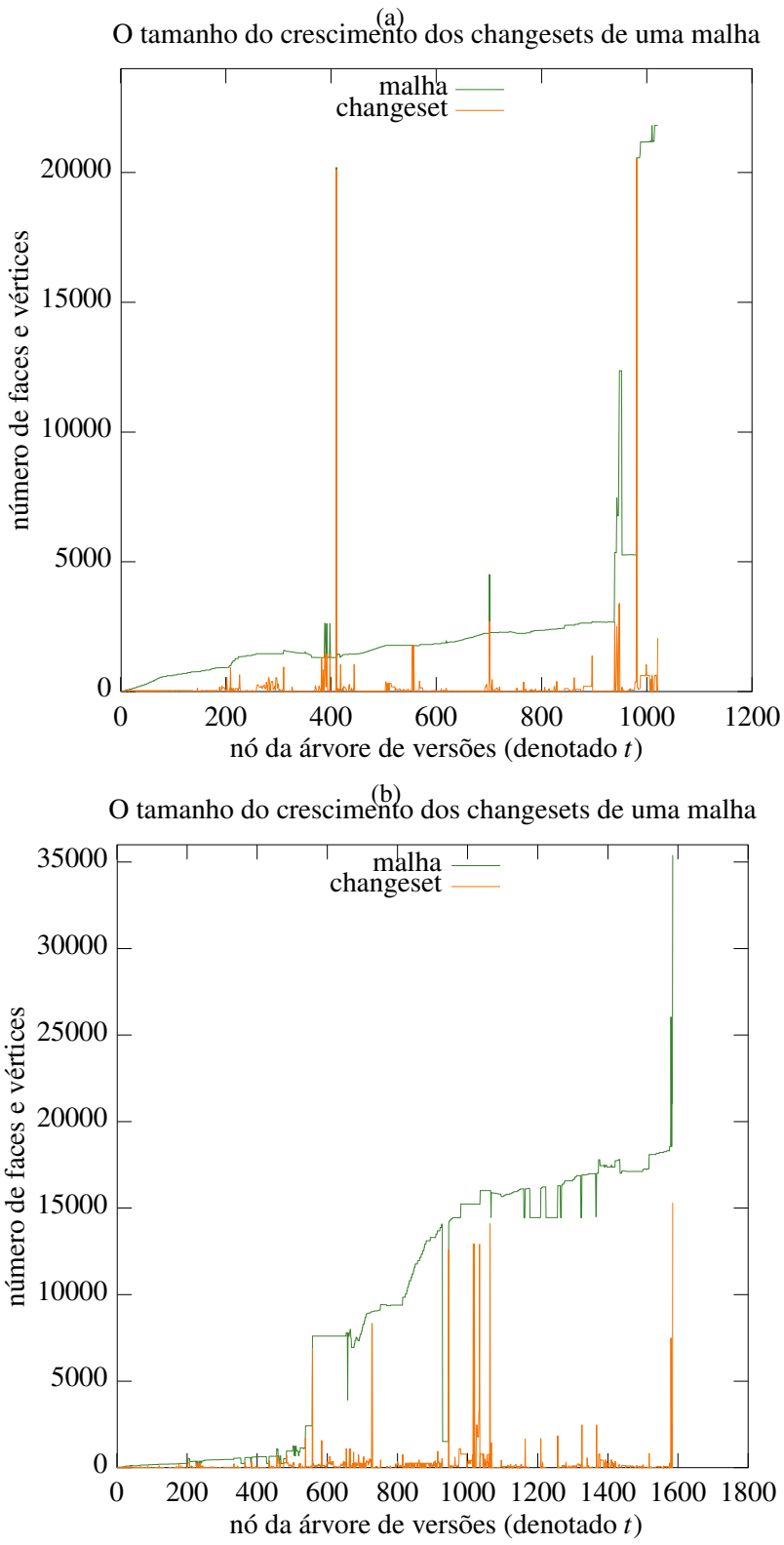
Alguns objetos na cena podem precisar ser removidos depois de um Desfazer Localizado. Por exemplo, o bigode da malha de Dom Quixote foi criado como um esfera fora do volume selecionado com uma Caixa Envoltória e, em seguida, a esfera foi arrastada e transformada na cena (Figura 4.8). Portanto, a esfera está na versão final, se todas as operações no bigode são desfeitas, já que a esfera é inserida fora do volume espacial ocupado pelo bigode. Isto ocorre pois apenas operações globais atualmente são capazes de afetar o posicionamento de uma Caixa Envoltória.

Para atravessar a árvore de versões completamente enquanto realizam-se operações geométricas, pode-se tomar muito tempo, e esta é a razão principal pela qual subárvores são importantes para SCVs 3D. O algoritmo desenvolvido foi implementado de forma paralela para aumentar sua velocidade atribuindo os processos efetuados sobre um ou dois nós da ACV por processador sempre que possível (Figura 4.18). A progressão da velocidade do algoritmo pode ser vista no Apêndice A.

Também foi observado que o tamanho dos Conjuntos de Mudanças é quase o mesmo ao longo de todas as Sequências de Construção (Figura 4.19), e que está relacionado ao tipo de operação realizada – razão pela qual a classificação das operações é um dos primeiros passos do algoritmo proposto. Picos da linha laranja na Figura 4.19 que tocam a linha verde são operações globais para malhas conexas, e é possível poupar tempo de computação evitando-as sempre que possível. Nós globais são permitidos nas subárvores submetidas ao Desfazer Localizado, pois tais nós são evitados (*skip*) devido a classificação prévia.

Apesar das limitações citadas, que promovem desafios para os trabalhos futuros, a técnica consegue lidar com outros tipos de estrutura além de malhas poligonais devido a análise geométrica de baixo nível que é realizada, além de desconsiderar operações cujo mapeamento não permite mapeamento válido de elementos. Como trabalho futuro, pode-se implementá-lo para outros tipos de modeladores *CAD* ou considerando operações de *Sculpting*.

Figura 4.19 – O tamanho de uma malha e seu *Changeset* ao longo dos nós de uma árvore de versões (Seção 4.7): a) Árvore de versões da malha do cavalo (Figura 4.4); b) Árvore de versões da malha de Dom Quixote (Figura 4.8).



Fonte: o autor.

5 CONCLUSÃO

Neste capítulo apresenta-se as implicações resultantes do trabalho desenvolvido, bem como propostas para trabalhos futuros.

Nesta tese foi introduzido uma funcionalidade aos sistemas de controle de versões para malhas poligonais (SCVs 3D), o Desfazer Localizado, bem como uma maneira para acompanhar a produção de uma determinada região, ou locus, da malha ao longo de sua construção por um artista através de subárvores especializadas. Até o presente momento desconhece-se qualquer trabalho similar existente com essa funcionalidade específica de Desfazer Localizado para malhas poligonais.

O método implementado é capaz de extrair das árvores de versão usadas pelos SCVs uma subárvore especializada baseada em Conjuntos de Mudanças ou *Changesets* que foram filtrados da árvore principal. Uma vez que obtém-se tal subárvore é possível gerar tutoriais usando Sequências de Construção desta, ou remover operações locais, i.e., operações que afetam um determinado locus (lugar) da malha, enquanto preserva-se outros loci para gerar novas versões da malha.

Os resultados apresentados nesta tese (Capítulo 4) confirmam que tal método, conforme descrito no Capítulo 3, é de fato viável. Embora existam desafios a serem superados, estes fazem parte da evolução natural do conhecimento na ciência em geometria computacional. Um exemplo destes desafios é como identificar erros do artista em uma malha poligonal. Já outros desafios exigem apenas que um tratamento dos dados prévio seja feito antes da execução do algoritmo para remoção de características indesejadas como faces duplicadas ou normais invertidas.

Pelo que já foi exposto, é possível afirmar ainda que a técnica apresentada é capaz de fornecer uma resposta bastante veloz. Contudo, ela ainda pode ser aperfeiçoada, caso os modeladores 3D ou SCVs 3D implementem alguns dos recursos sugeridos por esta tese. Por exemplo, o cálculo do Conjunto de Mudanças de forma particular pode ser realizado a cada operação de maneira transparente para um usuário pelo modelador ou SCV 3D, de modo que o tempo de execução do algoritmo se resuma completamente em testes de interseção dos Conjuntos de Mudanças com a Caixa Envoltória.

Observando-se os objetivos propostos (Capítulo 1), é possível concluir que todos eles foram alcançados e que a estratégia utilizada foi adequada para alcançar o efeito almejado, i.e., conseguiu-se extrair loci da malha para gerar tutoriais, e gerar versões novas de malhas

poligonais a partir de um desfazer especialista, o Desfazer Localizado.

O trabalho foi ainda analisado sobre critérios de paralelização e velocidade de acesso aos índices mapeados pelo uso das tabelas de dispersão, e, em ambos, comportou-se de forma satisfatória a um algoritmo que amplia as capacidades de um sistema de controle de versões para malhas. Alerta-se, contudo, que para casos de árvores pequenas ou malhas com poucos vértices o uso da técnica aqui descrita talvez não seja o mais indicado. A operação de deleção para esses casos mais simples ou mesmo a operação de desfazer comum pode ainda ser a mais indicada.

A comparação com uma técnica atual do estado da arte atesta que um método determinístico para cálculos geométricos parece ser o caminho mais indicado a se perseguir em SCVs para malhas poligonais ao invés de heurísticas que introduzem mais erros ao processo de modelagem, acumulando-os aos do artista, e dificultando ainda mais obter uma resposta adequada ao problema proposto.

5.1 Trabalhos Futuros

Como trabalho futuro, e de forma geral, novas propriedades dos Conjuntos de Mudanças podem ser exploradas, o método de seleção por Caixa Envoltória pode ser melhorado para uma geometria mais flexível, e novas aplicações para a subárvore especializada podem ser estudadas. Outras sugestões mais específicas seguem adiante.

A técnica implementada pode agir como um método de *debug* (análise e remoção de erros) para técnicas de geração de malhas. Qualquer algoritmo para geração de malhas poligonais pode usar subárvores especialistas desta técnica para determinar como regiões específicas foram geradas, filtrando passos indesejados e acompanhando o progresso da geração automática. Isto significa, claro, que estes algoritmos devem se comunicar com um SCV para poderem utilizarem o algoritmo e serem capazes de gerar múltiplos loci de interesse em uma malha.

O tempo para aplicar esta técnica ainda pode ser melhorado através do uso de outras técnicas de divisão espacial, de geração de chaves *hash*, e de testes de interseção. Mesmo que a tendência da velocidade dos computadores seja sempre a de crescer, nada sugere que o limite alcançado atualmente não possa ou deva ser melhorado.

Um outro trabalho possível é o uso desta técnica para mesclar árvores de versões usadas no processo de modelagem de diferentes malhas. Para isto, é necessária a atualização dos parâmetros de acordo com algum algoritmo de reconhecimento de $features$ (características) já que o reconhecimento de vértices em comum não deve ser usado como base para mapeamento

de modelos distintos (fora de uma mesma Sequência de Construção). Espera-se deste modo que se obtenha um algoritmo para geração procedural de malhas poligonais. Isto pode ser de grande valia para a indústria de entretenimento na geração de novos modelos de personagens combinando-se características de modelos já existentes conforme já é feito em 2D por meio de imagens.

Um algoritmo de reconhecimento de *features* ainda pode ser usado como tentativa de remover uma limitação do algoritmo atual, a remoção de nós globais, já que *features* tendem a não mudar de uma versão a outra da mesma malha. Outro método de visão computacional ou de geração de malhas por nuvem de pontos obtidas por scanners 3D ainda pode ser utilizado para aperfeiçoar o algortimo, já que ambas as áreas buscam reconhecer formas geométricas, e já que versões próximas de modelos em SCVs tendem a possui formas geométricas parecidas.

Durante o processo de modelagem, um artista pode possuir várias malhas em um cenário, ou usar de várias malhas como componentes de uma malha maior (desconexos entre si). Desta forma, uma possibilidade de trabalho futuro é extrair os componentes existentes na cena armazenada pela árvore de versões para a obtenção de uma floresta (conjuntos de elementos desconexos), de modo a acompanhar como determinados componentes foram feitos em uma malha ao invés de regiões determinadas.

Uma possibilidade final de aperfeiçoamento para o método é a de decompor as operações existentes em um ou mais nós sequenciais de forma a ser capaz de isolar perfeitamente apenas o locus afetado de interesse do usuário. Mesmo que uma operação afete mais que o desejado, decompô-la em operações menores pode resolver outra limitação atual da técnica isolando completamente as regiões de interesse.

REFERÊNCIAS

ALIENBRAIN. **Asset Management for Artists**. 2016. Disponível em: <http://www.alienbrain.com/features>. Acesso em: 30 de Julho de 2016.

ALLAIN, B.; FRANCO, J. S.; BOYER, E. An efficient volumetric framework for shape tracking. In: **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2015. p. 268–276. ISSN 1063-6919.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In: **Proceedings of the April 18-20, 1967, Spring Joint Computer Conference**. New York, NY, USA: ACM, 1967. (AFIPS '67 (Spring)), p. 483–485. Disponível em: <http://doi.acm.org/10.1145/1465482.1465560>.

APACHE SOFTWARE FOUNDATION. **Apache Subversion**. 2001. Disponível em: <http://subversion.apache.org>. Acesso em: 31 de Julho de 2016.

AUTODESK, I. **API MEL/Python**. 2011. Disponível em: <http://download.autodesk.com/us/maya/2011help/Commands/cat_General.html>. Acesso em: 30 de Julho de 2016.

AUTODESK, I. **Autodesk Maya**. 2016. Disponível em: <http://usa.autodesk.com>. Acesso em: 30 de Julho de 2016.

AUTODESK, I. **MEL, Maya Embedded Language**. 2016. Disponível em: <http://download.autodesk.com/global/docs/maya2014/en_us/index.html?url=files/Background_MEL_for_programmers.htm,topicNumber=d30e788742>. Acesso em: 30 de Julho de 2016.

BA, M. L.; ABDESSALEM, T.; SENELLART, P. Uncertain version control in open collaborative editing of tree-structured documents. In: **Proceedings of the 2013 ACM Symposium on Document Engineering**. New York, NY, USA: ACM, 2013. (DocEng '13), p. 27–36. ISBN 978-1-4503-1789-4.

BADAWY, W. M.; AREF, W. G. On local heuristics to speed up polygon-polygon intersection tests. In: **In ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems**. [S.l.: s.n.], 1999. p. 97–102.

BAUDRIT, A. **Le tutorat. Richesses d'une méthode pédagogique**. [S.l.]: De Boeck, 2003. ISBN 978-2804142537.

BAVOIL, L.; CALLAHAN, S. P.; CROSSNO, P. J.; FREIRE, J.; SCHEIDEGGER, C. E.; SILVA, C. T.; VO, H. T. Vistrails: Enabling interactive multiple-view visualizations. In: IEEE. **VIS 05. IEEE Visualization, 2005.** [S.l.], 2005. p. 135–142.

BLENDER. 2016. Disponível em: <http://www.blender.org>. Acesso em: 30 de Julho de 2016.

BRUNET, P.; NAVAZO, I. Solid representation and operation using extended octrees. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 9, n. 2, p. 170–197, Abril 1990. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/78956.78959>.

BRUTZMAN, D.; DALY, L. **X3D: Extensible 3D Graphics for Web Authors**. 2004. Disponível em: <http://x3dgraphics.com/>. Acesso em: 30 de Julho de 2016.

CAGNIART, C.; BOYER, E.; ILIC, S. Free-form mesh tracking: A patch-based approach. In: **Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on**. [S.l.: s.n.], 2010. p. 1339–1346. ISSN 1063-6919.

CATMULL, E.; CLARK, J. Recursively generated b-spline surfaces on arbitrary topological meshes. **Computer-Aided Design**, v. 10, n. 6, p. 350 – 355, 1978. ISSN 0010-4485. Disponível em: <http://www.sciencedirect.com/science/article/pii/0010448578901100>.

CHAN, C.-Y.; IOANNIDIS, Y. E. Bitmap index design and evaluation. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 27, n. 2, p. 355–366, Junho 1998. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/276305.276336>.

CHANG, C.-T.; GORISSEN, B.; MELCHIOR, S. Fast oriented bounding box optimization on the rotation group so(3,$\mathbb{R}$). **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 30, n. 5, p. 122:1–122:16, Outubro 2011. ISSN 0730-0301.

CHEN, H.-T.; GROSSMAN, T.; SCHMIDT, R.; HARTMANN, B.; FITZMAURICE, G.; AGRAWALA, M. History assisted view authoring for 3d models. **ACM Conference on Human Factors in Computing Systems**, 2014.

CHEN, H.-T.; WEI, L.-Y.; CHANG, C.-F. Nonlinear revision control for images. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 30, n. 4, p. 105:1–105:10, Julho 2011. ISSN 0730-0301.

CHEN, H.-T.; WEI, L.-Y.; HARTMANN, B.; AGRAWALA, M. Data-driven adaptive history for image editing. In: **Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games**. New York, NY, USA: ACM, 2016. (I3D '16), p. 103–111. ISBN 978-1-4503-4043-4. Disponível em: <http://doi.acm.org/10.1145/2856400.2856417>.

CHENG, Y.; CAI, X.; HE, F.; ZHANG, D. A selective undo/redo method in 3d collaborative modeling environment. In: **Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on**. [S.l.: s.n.], 2012. p. 102–108.

CHENG, Y.; HE, F.; JING, S.; HUANG, Z. An multiuser undo/redo method for replicated collaborative modeling systems. In: **Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design**. Washington, DC, USA: IEEE Computer Society, 2009. (CSCWD '09), p. 185–190. ISBN 978-1-4244-3534-0. Disponível em: <http://dx.doi.org/10.1109/CSCWD.2009.4968056>.

CHENG, Y.; HE, F.; XU, B.; HAN, S.; CAI, X.; CHEN, Y. A multi-user selective undo/redo approach for collaborative CAD systems. **Journal of Computational Design and Engineering**, v. 1, n. 2, p. 103 – 115, 2014. ISSN 2288-4300. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2288430014500164>.

CHOU, H.-T.; KIM, W. A unifying framework for version control in a CAD environment. In: **Proceedings of the 12th International Conference on Very Large Data Bases**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986. (VLDB '86), p. 336–344. ISBN 0-934613-18-4.

CORBET, J.; RUBINI, A.; KROAH-HARTMAN, G. **Linux Device Drivers, 3rd Edition**. [S.l.]: O'Reilly Media, Inc., 2005. ISBN 0596005903.

CORMEN, T. H.; STEIN, C.; RIVEST, R. L.; LEISERSON, C. E. **Introduction to Algorithms**. 2nd. ed. [S.l.]: McGraw-Hill Higher Education, 2001. ISBN 0070131511.

DEGTYAREV, Y.; DAVIDSON, P.; FANELLO, S.; KOWDLE, A.; ESCOLANO, S. O.; RHEMANN, C.; KIM, D.; TAYLOR, J.; DOU, M.; KHAMIS, S.; KOHLI, P.; TANKOVICH, V.; IZADI, S. Fusion4d: Real-time performance capture of challenging scenes. In: . SIGGRAPH, 2016. Disponível em: <https://www.microsoft.com/en-us/research/publication/fusion4d-real-time-performance-capture-of-challenging-scenes/>.

DENNING, J. D. **ModFlows: Methods for Studying and Managing Mesh Editing Workflows**. Tese (Doutorado) — Dartmouth College, Computer Science, Hanover, NH, Junho 2014. Disponível em: <http://gradworks.umi.com/36/33/3633485.html>.

DENNING, J. D.; KERR, W. B.; PELLACINI, F. Meshflow: Interactive visualization of mesh construction sequences. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 30, n. 4, p. 66:1–66:8, Julho 2011. ISSN 0730-0301.

DENNING, J. D.; PELLACINI, F. Meshgit: Diffing and merging meshes for polygonal modeling. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 32, n. 4, p. 35:1–35:10, Julho 2013. ISSN 0730-0301.

DENNING, J. D.; TIBALDO, V.; PELLACINI, F. 3dflow: Continuous summarization of mesh editing workflows. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 34, n. 4, p. 140:1–140:9, Julho 2015. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/2766936>.

DFKI; UNIVERSITY, C. L. of the S.; INSTITUTE the I. V. C. **XML3D**. 2011. Disponível em: <http://xml3d.org>. Acesso em: 30 de Julho de 2016.

DOBOŠ, J. **Management and Visualisation of Non-linear History of Polygonal 3D Models**. Tese (Doutorado) — UCL (University College London), London, 2015. Disponível em: <http://discovery.ucl.ac.uk/1461400/>.

DOBOŠ, J. **3D Repo: 3D Version Control System**. 2016. Disponível em: <http://3drepo.org/>. Acesso em: 30 de Julho de 2016.

DOBOŠ, J.; MITRA, N. J.; STEED, A. 3d timeline: Reverse engineering of a part-based provenance from consecutive 3d models. **Computer Graphics Forum (Special issue of Eurographics 2014)**, 2014.

DOBOŠ, J.; SONS, K.; RUBINSTEIN, D.; SLUSALLEK, P.; STEED, A. Xml3drepo: A rest api for version controlled 3d assets on the web. In: **Proceedings of the 18th International Conference on 3D Web Technology**. New York, NY, USA: ACM, 2013. (Web3D '13), p. 47–55. ISBN 978-1-4503-2133-4.

DOBOŠ, J.; STEED, A. 3d diff: An interactive approach to mesh differencing and conflict resolution. In: **ACM SIGGRAPH 2012 Talks**. New York, NY, USA: ACM, 2012. (SIGGRAPH '12), p. 15:1–15:1. ISBN 978-1-4503-1683-5.

DOBOŠ, J.; STEED, A. **Revision Control Database for 3D Assets**. 2012. Disponível em: <http://web.archive.org/web/20150426152241/http://www.cs.ucl.ac.uk/research/research_notes/>. Acesso em: Relatório Técnico. 30 de Julho de 2016.

DREAMWORKS. **Shrek: o Filme**. 2001. Disponível em: <http://www.d.umn.edu/cla/faculty/troufs/anth4616/images/Shrek_3D.jpg>. Acesso em: 30 de Julho de 2016.

DU, Q.; FABER, V.; GUNZBURGER, M. Centroidal voronoi tessellations: Applications and algorithms. **SIAM Review**, v. 41, n. 4, p. 637–676, 1999. Disponível em: <http://dx.doi.org/10.1137/S0036144599352836>.

ELYAMANY, H. F.; ELAWADY, M. E.; ABDELHAY, E. E.; KHALIL, A. I. **A Web Service-Based Framework for an Online 3D Model Viewer**. 2012. Disponível em: <https://fedcsis.org/proceedings/2012/pliks/200.pdf>. Acesso em: 30 de Julho de 2016.

ERICSON, C. **Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN 1558607323.

FOLIO. 2016. Disponível em: <http://folioformac.com>. Acesso em: 30 de Julho de 2016.

FREIRE, J.; KOOP, D.; CHIRIGATI, F. S.; SILVA, C. T. Reproducibility using vistrails. **Implementing Reproducible Research**, CRC Press, v. 33, 2014.

GIRARD, A.; BELLIK, Y.; AUVRAY, M.; AMMI, M. Visuo-haptic tool for collaborative adjustment of selections. In: **Haptic and Audio Interaction Design**. [S.l.]: Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 7989). p. 40–49. ISBN 978-3-642-41067-3.

GITHUB. **Rendering and diffing images**. 2016. Disponível em: <https://help.github.com/articles/rendering-and-diffing-images/>. Acesso em: 30 de Julho de 2016.

GRABLER, F.; AGRAWALA, M.; LI, W.; DONTCHEVA, M.; IGARASHI, T. Generating photo manipulation tutorials by demonstration. In: **ACM SIGGRAPH 2009 Papers**. New York, NY, USA: ACM, 2009. (SIGGRAPH '09), p. 66:1–66:9. ISBN 978-1-60558-726-4.

GRAY, J. The transaction concept: Virtues and limitations. **Proceedings of the 7th International Conference on Very Large Databases**, Tandem Computers Incorporated, Cupertino, CA, USA, p. 144–154, 1981.

GUÉZIEC, A.; PENNEC, X.; AYACHE, N. Medical Image Registration using Geometric Hashing. **IEEE Computational Science and Engineering**, Institute of Electrical and Electronics Engineers (IEEE), v. 4, n. 4, p. 29–41, 1997. Disponível em: <https://hal.inria.fr/inria-00615068>.

HAKIM, A. **3D model of Stan Lee**. 2012. Disponível em: <http://ceiga.co.uk/blog/stan-the-man/>. Acesso em: 17 de Julho de 2016.

HICKSON, I.; BERJON, R.; FAULKNER, S.; NAVARA, E. D.; O'CONNOR, E.; PFEIFFER, S. **HTML5**. 2014. Disponível em: <http://www.w3.org/TR/html5/>. Acesso em: 30 de Julho de 2016.

HU, S. M.; XU, K.; MA, L.-Q.; LIU, B.; JIANG, B.-Y.; WANG, J. Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 32, n. 6, p. 194:1–194:11, Novembro 2013. ISSN 0730-0301.

HUBERT, I. **Tears of Steel, Mango Open Movie Project**. 2013. Disponível em: <http://mango.blender.org/>. Acesso em: 30 de Julho de 2016.

JADEJA, A.; MEHTA, R.; SHARMA, D. New era of teaching learning : 3d marker based augmented reality. **International Journal of Information Sciences and Techniques (IJIST)**, v. 6, n. 1/2, p. 81–88, 2016.

JAIN, A.; THORMÄHLEN, T.; RITSCHEL, T.; SEIDEL, H.-P. Exploring shape variations by 3d-model decomposition and part-based recombination. **Comp. Graph. Forum (Proc. Eurographics 2012)**, v. 31, n. 2, 2012.

KAWASAKI, Y.; IGARASHI, T. Regional undo for spreadsheets. In: **Proc. ACM Symposium on User Interface Software and Technology**. [S.l.: s.n.], 2004.

KHRONOS, G. **OpenGL ES 2.0 for the Web**. 2014. Disponível em: <https://www.khronos.org/webgl/>. Acesso em: 30 de Julho de 2016.

LEFEBVRE, S.; HOPPE, H. Perfect spatial hashing. In: **ACM SIGGRAPH 2006 Papers**. New York, NY, USA: ACM, 2006. (SIGGRAPH '06), p. 579–588. ISBN 1-59593-364-6. Disponível em: <http://doi.acm.org/10.1145/1179352.1141926>.

LI, R.; LI, D. A regional undo mechanism for text editing. **The 5th International Workshop on Collaborative Editing Systems**, 2003.

LORD, T. **GNU arch**. 2005. Disponível em: <https://www.gnu.org/software/gnu-arch/>. Acesso em: 31 de Julho de 2016.

MA, R.; E., S.; D, P.; R, J.; D, F.; D, L. Demonstration-based training: a review of instructional features. **Human Factors and Ergonomics**, v. 52, n. 5, p. 596–609, Outubro 2010.

MACKALL, M. **Mercurial Source Control Management**. 2005. Disponível em: <https://www.mercurial-scm.org/>. Acesso em: 31 de Julho de 2016.

MAKERBOT. **MakeBot Digitizer**. 2016. Disponível em: <https://www.makerbot.com>. Acesso em: 17 de Julho de 2016.

MENG, C.; YASUE, M.; IMAMIYA, A.; MAO, X. Visualizing histories for selective undo and redo. In: **Proceedings of the Third Asian Pacific Computer and Human Interaction**. Washington, DC, USA: IEEE Computer Society, 1998. (APCHI '98), p. 459–. ISBN 0-8186-8347-3. Disponível em: <http://dl.acm.org/citation.cfm?id=786112.786260>.

MIAN, A. S.; BENNAMOUN, M.; OWENS, R. Three-dimensional model-based object recognition and segmentation in cluttered scenes. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 28, n. 10, p. 1584–1601, Outubro 2006. ISSN 0162-8828.

MYERS, B. A.; LAI, A.; LE, T. M.; YOON, Y.; FAULRING, A.; BRANDT, J. Selective undo support for painting applications. In: **Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2015. (CHI '15), p. 4227–4236. ISBN 978-1-4503-3145-6. Disponível em: <http://doi.acm.org/10.1145/2702123.2702543>.

NAKAMURA, T.; IGARASHI, T. An application-independent system for visualizing user operation history. In: **Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology**. [S.l.]: ACM, 2008. (UIST '08), p. 23–32. ISBN 978-1-59593-975-3.

NASA. **What Wavelength Goes With a Color?** 2016. Disponível em: <http://science-edu.larc.nasa.gov/EDDOCS/Wavelengths_for_Colors.html>. Acesso em: 30 de Julho de 2016.

NEGARA, S.; VAKILIAN, M.; CHEN, N.; JOHNSON, R. E.; DIG, D. Is it dangerous to use version control histories to study source code evolution? In: **Proceedings of the 26th European Conference on Object-Oriented Programming**. Berlin, Heidelberg: Springer-Verlag, 2012. (ECOOP'12), p. 79–103. ISBN 978-3-642-31056-0.

NUSS, J. E. **Assessing User Expectations of Undo in a Multi-User CAD Environment**. Dissertação (Mestrado) — Brigham Young University, Provo, 2016.

NYAMSUREN, P.; LEE, S.-H.; KIM, S. A web-based revision control framework for 3d CAD model data. **International Journal of Precision Engineering and Manufacturing**, Springer Berlin Heidelberg, v. 14, n. 10, p. 1797–1803, 2013. ISSN 2234-7593.

N.Y.U. **Vistrails**. 2005. Disponível em: <http://www.vistrails.org/>. Acesso em: 31 de Julho de 2016.

N.Y.U. **VisTrails Provenance Explorer for Maya**. 2009. Disponível em: <http://web.archive.org/web/20131211010210/http://vistrails.com/maya.html>. Acesso em: 30 de Julho de 2016.

OSADA, R.; FUNKHOUSER, T.; CHAZELLE, B.; DOBKIN, D. Matching 3d models with shape distributions. In: **Shape Modeling and Applications, SMI 2001 International Conference on.** [S.l.: s.n.], 2001. p. 154–166.

PAPADIMITRIOU, C. H. Np-completeness: A retrospective. In: ____. **Automata, Languages and Programming: 24th International Colloquium, ICALP '97 Bologna, Italy, Julho 7–11, 1997 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. p. 2–6. ISBN 978-3-540-69194-5. Disponível em: <http://dx.doi.org/10.1007/3-540-63165-8_160>.

PIXELAPSE. **Visual Version Control**. 2016. Disponível em: <https://www.pixelapse.com>. Acesso em: 30 de Julho de 2016.

PONZANELLI, L.; BAVOTA, G.; MOCCI, A.; PENTA, M. D.; OLIVETO, R.; HASAN, M.; RUSSO, B.; HAIDUC, S.; LANZA, M. Too long; didn't watch!: Extracting relevant fragments from software development video tutorials. In: **Proceedings of the 38th International Conference on Software Engineering**. New York, NY, USA: ACM, 2016. (ICSE '16), p. 261–272. ISBN 978-1-4503-3900-1. Disponível em: <http://doi.acm.org/10.1145/2884781.2884824>.

POOL, M. **Integrals and derivatives**. 2004. Disponível em: <http://archive.is/ctWmx>. Acesso em: 30 de Julho de 2016.

PUJOL, S.; BALDWIN, M.; NASSIRI, J.; KIKINIS, R.; SHAFFER, K. Using 3d modeling techniques to enhance teaching of difficult anatomical concepts. **Academic Radiology**, v. 23, n. 4, p. 507–516, 2016.

RAYMOND, E. S. **Understanding Version-Control Systems**. 1992. Disponível em: <http://www.catb.org/esr/writings/version-control/version-control.html>. Acesso em: 31 de Julho de 2016.

ROCHKIND, M. J. The source code control system. **IEEE Transactions on Software Engineering**, SE-1, n. 4, Dezembro 1975.

RUPARELIA, N. B. The history of version control. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 35, n. 1, p. 5–9, Janeiro 2010. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/1668862.1668876>.

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4th. ed. [S.l.]: Addison-Wesley Professional, 2011. ISBN 978-0321573513.

SEIFRIED, T.; RENDL, C.; HALLER, M.; SCOTT, S. Regional undo/redo techniques for large interactive surfaces. In: **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2012. (CHI '12), p. 2855–2864. ISBN 978-1-4503-1015-4. Disponível em: <http://doi.acm.org/10.1145/2207676.2208690>.

SKALNIK, M. **3D file diff and viewer**. 2013. Disponível em: <https://github.com/blog/1633-3d-file-diffs>. Acesso em: 30 de Julho de 2016.

SOCOLOFSKY, T.; KALE, C. **A TCP/IP Tutorial**. 1991. Disponível em: <https://tools.ietf.org/html/rfc1180>. Acesso em: 31 de Julho de 2016.

STROZZI, C. **NoSQL: a non-SQL RDBMS**. 1998. Disponível em: <http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home\%20Page>. Acesso em: 30 de Julho de 2016.

SU, S. L.; PARIS, S.; ALIAGA, F.; SCULL, C.; JOHNSON, S.; DURAND, F. **Interactive Visual Histories for Vector Graphics**. Cambridge, MA, 2009.

THE DOCUMENT FOUNDATION. **Libreoffice - A melhor suite office livre**. 2016. Disponível em: <https://pt-br.libreoffice.org>. Acesso em: 31 de Julho de 2016.

TICHY, W. F. **GNU RCS**. 1982. Disponível em: <https://www.gnu.org/software/rcs/>. Acesso em: 31 de Julho de 2016.

TICHY, W. F. **RCS - GNU Project - Free Software Foundation**. 1982. Disponível em: <http://www.gnu.org/software/rcs/>. Acesso em: 31 de Julho de 2016.

TORVALDS, L. **Git –fast-version-control**. 2011. Disponível em: <http://git-scm.com/>. Acesso em: 31 de Julho de 2016.

WIKIPÉDIA. **Comparison of version control software**. 2016. Disponível em: <https://en.wikipedia.org/wiki/Comparison_of_version_control_software>. Acesso em: 31 de Julho de 2016.

WINTER, K. **Aparição do holograma de Michael Jackson no Billboard Music Awards**. 2014. Disponível em: <http://www.eonline.com/news/542973/michael-jackson-s-hologram-takes-the-2014-billboard-music-awards-stage-see-the-pic>. Acesso em: 17 de Julho de 2016.

WOLFSON, H. J.; RIGOUTSOS, I. Geometric hashing: An overview. **Computing in Science and Engineering**, IEEE Computer Society, Los Alamitos, CA, USA, v. 4, n. 4, p. 10–21, 1997. ISSN 1070-9924.

YOON, Y. S.; MYERS, B. A. Supporting selective undo in a code editor. In: **Proceedings of the 37th International Conference on Software Engineering - Volume 1**. Piscataway, NJ, USA: IEEE Press, 2015. (ICSE '15), p. 223–233. ISBN 978-1-4799-1934-5. Disponível em: <http://dl.acm.org/citation.cfm?id=2818754.2818784>.

ZEISS, B. **subversion 1.0 is released**. 2004. Disponível em: <http://lwn.net/Articles/72498/>. Acesso em: 31 de Julho de 2016.

ZHOU, C.; IMAMIYA, A. Object-based nonlinear undo model. In: **Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International**. [S.l.: s.n.], 1997. p. 50–55. ISSN 0730-3157.

APÊNDICE A – PROGRESSÃO DA VELOCIDADE

A.1 Tempo do algoritmo em Python

Este tempo foi obtido após o primeiro *breakthrough* do algoritmo ao implementar tabelas de dispersão.

Tabela A.1 – Análise do algoritmo. O: ACV original; E: árvore enriquecida; S: árvore especializada; e T: subárvore especializada (Figura 3.2).

Malha	Seleção determinada	No. de nós: subárvore / ACV	Tempo		
			$O \to E$	$E \to S$	$S \to T$
Espaçonave	Estabilizador	8 / 92	<1s	61s	<1s
	Asa Esq.	16 / 92	1.3 s	65 s	<1s
Cavalo	Orelhas	147 / 1196	11.1s	48m 14s	<1s
	Cabeça	353 / 1196	11.3s	52m 40s	1.5s
	Patas Front.	104 / 1196	11.4s	50m 15s	1.5s
	Todas Patas	210 / 1196	10.1s	54m 26s	<1s
Dom Quixote	Bigode	177 / 1648	24.3s	283m 12s	1.2s
	Cavanhaque	144 / 1648	24.0s	283m 46s	<1s
	Mão Esq.	110 / 1648	23.1s	286m 4s	1.3s
	Pernas	134 / 1648	23.1s	321m 44s	1.2s

Fonte: o autor.

A.2 Tempo do algoritmo em Python com subdivisão do espaço

Tabela A.2 – Análise do algoritmo. O: ACV original; E: árvore enriquecida; S: árvore especializada; e T: subárvore especializada (Figura 3.2).

Malha	Seleção determinada	No. de nós: subárvore / ACV	Tempo		
			$O \to E$	$E \to S$	$S \to T$
Espaçonave	Estabilizador	8 / 92	<1s	15s	<1s
	Asa Esq.	16 / 92	1.3s	53s	<1s
Cavalo	Orelhas	147 / 1196	11.1s	15m 17s	<1s
	Cabeça	353 / 1196	11.3s	26m 36s	1.5s
	Patas Front.	104 / 1196	11.4s	16m 58s	1.5s
	Todas Patas	210 / 1196	10.1s	23m 58s	<1s
Dom Quixote	Bigode	177 / 1648	24.3s	63m 34s	1.2s
	Cavanhaque	144 / 1648	24.0s	61m 21s	<1s
	Mão Esq.	110 / 1648	23.1s	59m 47s	1.3s
	Pernas	134 / 1648	23.1s	150m 26s	1.2s

Fonte: o autor.

A.3 Tempo do algoritmo em C com subdivisão do espaço

O código foi todo reescrito em C oferecendo um ganho considerável de velocidade. Neste ponto, a malha da Espaçonave já obtém o resultado exibido na Tese.

Tabela A.3 – Análise do algoritmo. O: ACV original; E: árvore enriquecida; S: árvore especializada; e T: subárvore especializada (Figura 3.2).

Malha	Seleção determinada	No. de nós: subárvore / ACV	Tempo		
			$O \rightarrow E$	$E \rightarrow S$	$S \rightarrow T$
Espaçonave	Estabilizador	8 / 92	<1s	<1s	<1s
	Asa Esq.	16 / 92	<1s	<1s	<1s
Cavalo	Orelhas	147 / 1196	11.1s	27s	<1s
	Cabeça	353 / 1196	11.3s	23s	1.5s
	Patas Front.	104 / 1196	11.4s	18s	1.5s
	Todas Patas	210 / 1196	10.1s	27s	<1s
Dom Quixote	Bigode	177 / 1648	24.3s	1m 20s	1.2s
	Cavanhaque	144 / 1648	24.0s	1m 19s	<1s
	Mão Esq.	110 / 1648	23.1s	1m 14s	1.3s
	Pernas	134 / 1648	23.1s	2m 41s	1.2s

Fonte: o autor.

A.4 Tempo do algoritmo em C com subdivisão do espaço e paralelismo

Foi adicionado ao código a biblioteca OPENMP para efetuar o paralelismo. Após este avanço, a tabela exibida na Tese foi obtida por ajustes e correções no código melhorando o uso da memória.

Tabela A.4 – Análise do algoritmo. O: ACV original; E: árvore enriquecida; S: árvore especializada; e T: subárvore especializada (Figura 3.2).

Malha	Seleção determinada	No. de nós: subárvore / ACV	Tempo		
			$O \rightarrow E$	$E \rightarrow S$	$S \rightarrow T$
Espaçonave	Estabilizador	8 / 92	<1s	<1s	<1s
	Asa Esq.	16 / 92	<1s	<1s	<1s
Cavalo	Orelhas	147 / 1196	11.1s	3s	<1s
	Cabeça	353 / 1196	11.3s	5s	<1s
	Patas Front.	104 / 1196	11.4s	4s	<1s
	Todas Patas	210 / 1196	10.1s	6s	<1s
Dom Quixote	Bigode	177 / 1648	24.3s	12s	<1s
	Cavanhaque	144 / 1648	24.0s	12s	<1s
	Mão Esq.	110 / 1648	23.1s	12s	<1s
	Pernas	134 / 1648	23.1s	31s	<1s

Fonte: o autor.

ANEXO A – EXEMPLO DE FUNÇÃO API DO MAYA

Esta função está contida no arquivo *data* referenciada pelo nó 1111 da malha do Cavalo (Uma operação de escala local). Aqui ela é exibida em formato abreviado, porém pode-se obervar uma lista de faces afetadas, como informações sobre câmera e seleção.

Código-fonte 3 – Função MEL salva pelo plugin do VT

```
1 changenodeattrU setAttr "|polySurface4|polySurfaceShape5.pt
    [2496]" -type "float3" 0.020441758 0 0 ;
2 |polySurface4|polySurfaceShape5|polySurface4|
    polySurfaceShape5(|polySurface4|polySurfaceShape5.pt
    [2496]pt[2496]changenodeattrV  setAttr "|polySurface4|
    polySurfaceShape5.pt[2405]" -type "float3" 0.0083863288
    0 0 ;...;
3 |polySurface4|polySurfaceShape5|polySurface4|
    polySurfaceShape5(|polySurface4|polySurfaceShape5.pt
    [2405]pt[2405]changenodeattrU  setAttr "|polySurface4|
    polySurfaceShape5.pt[2398]" -type "float3" 0.011139481 0
     0 ;selectType  -alo 1 -alc 0; selectType -ocm -alo 1 -
    alc 0; selectType  -cv 1 -hl 1 -se 1 -sf 1 -iso 1 -sme 1
     -smf 1 -lp 1 -pr 1 -spc 1 -eg 0 -fc 0 -pe 0 -pfe 0 -pf
    0 -puv 0 -smp 1 -v 1 -c 1 -pv 1 -ac 1 -ak 1 -ait 1 -aot
    1 -abd 1; selectType -ocm -smp 0 -v 0 -eg 0 -fc 0 -pv 0
    -pe 0 -pfe 0 -pf 0 -puv 0 -c 1 -ac 1 -ak 1 -ait 1 -aot 1
     -abd 1; display$editorName = "modelPanel1";
                modelEditor -e
4                -camera "top"
5                -useInteractiveMode 0
6                -displayLights "default"
7                -displayAppearance "smoothShaded"
8                -activeOnly 0
9                -...
```

ANEXO B – COMANDOS DA MEL PARA OS QUAIS O ALGORITMO FOI TESTADO

A lista de comandos da API para a versão 2011 do Maya (AUTODESK, 2011).

B.1 Edição e diversos

a) about;
b) addPrefixToName;
c) affectedNet;
d) affects;
e) align;
f) applicationName;
g) bakePartialHistory;
h) color;
i) colorIndex;
j) commandPort;
k) container;
l) containerAssignMaterial;
m) containerAssignTemplate;
n) containerAutobind;
o) containerAutobind;
p) containerAutopublishRoot;
q) containerBind;
r) containerCreateBindingSet;
s) containerDefaultBindingSet;
t) containerProxy;
u) containerPublish;
v) containerRemoveBindingSet;
w) containerRmbMenu;
x) containerTemplate;
y) containerView;
z) createDisplayLayer;
aa) delete;

ab) deleteAllContainers;
ac) distanceDimension;
ad) doPublishNode;
ae) duplicate;
af) editDisplayLayerGlobals;
ag) editDisplayLayerMembers;
ah) exactWorldBoundingBox;
ai) firstParentOf;
aj) getApplicationVersionAsFloat;
ak) getCurrentContainer;
al) group;
am) instance;
an) instanceable;
ao) instancer;
ap) isParentOf;
aq) isSameObject;
ar) license;
as) listSets;
at) listTransforms;
au) listUnselected;
av) lockContainer;
aw) lockNode;
ax) longNameOf;
ay) ls;
az) lsType;
ba) makeIdentity;
bb) makeLive;
bc) makePaintable;
bd) move;
be) obsoleteProc;
bf) paramDimension;
bg) paramLocator;

bh) parent;
bi) partition;
bj) performanceOptions;
bk) pixelMove;
bl) plugAttr;
bm) plugMultiAttrs;
bn) plugNode;
bo) plugNodeStripped;
bp) publishAnchorNodes;
bq) publishContainerConnections;
br) quit;
bs) rename;
bt) renameSelectionList;
bu) reorder;
bv) reorderContainer;
bw) rootOf;
bx) rotate;
by) scale;
bz) scaleComponents;
ca) sets;
cb) shapeCompare;
cc) shortNameOf;
cd) snapMode;
ce) snapTogetherCtx;
cf) spaceLocator;
cg) stripPrefixFromName;
ch) threadCount;
ci) toggleAxis;
cj) transformCompare;
ck) transformLimits;
cl) ungroup;
cm) upAxis; e

cn) xform.

B.2 Atributos

a) addAttr;
b) aliasAttr;
c) attributeExists;
d) attributeInfo;
e) attributeName;
f) attributeQuery;
g) connectAttr;
h) connectionInfo;
i) copyAttr;
j) createNode;
k) cycleCheck;
l) deleteAttr;
m) disconnectAttr;
n) editAttrLimits;
o) getAttr;
p) getClassification;
q) inheritTransform;
r) isConnected;
s) isDirty;
t) listAttr;
u) listConnections;
v) listHistory;
w) listNodeTypes;
x) listRelatives;
y) nodeCast;
z) nodeType;
aa) objectCenter;
ab) objectLayer;
ac) objectType;

ad) objExists;
ae) relationship;
af) removeMultiInstance;
ag) renameAttr;
ah) setAttr;
ai) setCustomAttrEnumResource;
aj) setCustomAttrNiceNameResource; e
ak) setState.

B.3 Seleção

a) hilite;
b) isolateSelect;
c) pickWalk;
d) select;
e) selectedNodes;
f) selectKey;
g) selectMode;
h) selectPref;
i) selectPriority;
j) selectType;
k) softSelect; e
l) symmetricModelling.